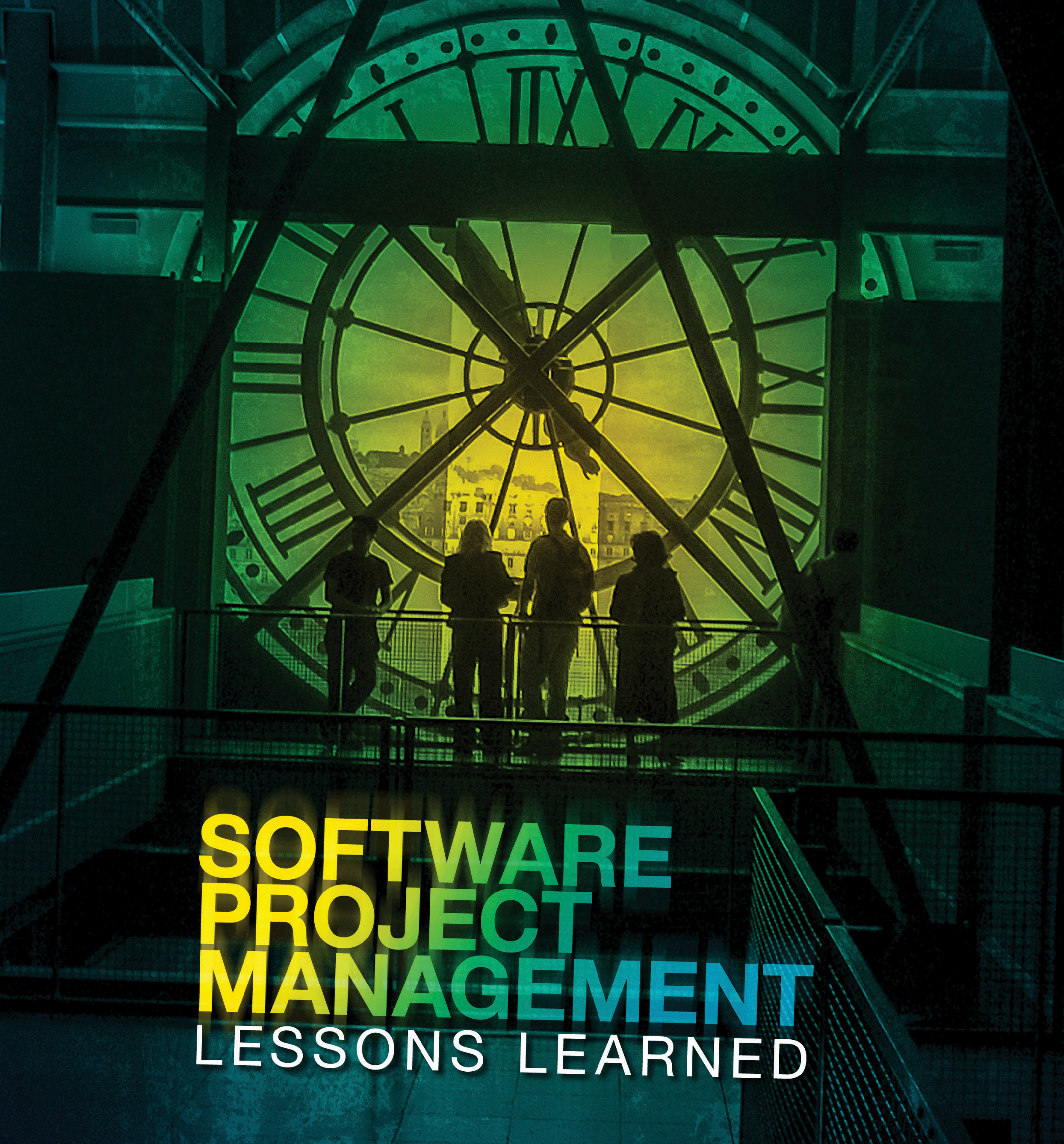
CrossTalk

Jan / Feb 2013 **The Journal of Defense Software Engineering** Vol. 26 No. 1



SOFTWARE PROJECT MANAGEMENT

LESSONS LEARNED

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **FEB 2013**	2. REPORT TYPE	3. DATES COVERED **00-01-2013 to 00-02-2013**

4. TITLE AND SUBTITLE **CrossTalk, The Journal of Defense Software Engineering. Volume 26, Number 1. Jan/Feb 2013**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **517 SMXS MXDEA,6022 Fir Ave Bldg 1238,Hill AFB,UT,84056-5820**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

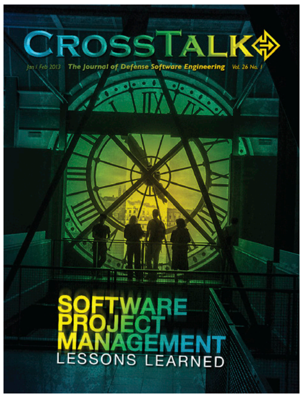
15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**40**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

TABLE OF CONTENTS

Departments



Cover Design by Kent Bingham

Software Project Management - Lessons Learned

CrossTalk

NAVAIR Jeff Schwalb
DHS Joe Jarzombek
309 SMXG Karl Rogers

Publisher Justin T. Hill
Advisor Kasey Thompson
Article Coordinator Lynne Wade
Managing Director Tracy Stauder
Managing Editor Brandon Ellis
Associate Editor Colin Kelly
Art Director Kevin Kiernan

Phone 801-775-5555
E-mail stsc.customerservice@hill.af.mil
CrossTalk Online www.crosstalkonline.org

CrossTalk, The Journal of Defense Software Engineering is co-sponsored by the U.S. Navy (USN); U.S. Air Force (USAF); and the U.S. Department of Homeland Defense (DHS). USN co-sponsor: Naval Air Systems Command. USAF co-sponsor: Ogden-ALC 309 SMXG. DHS co-sponsor: National Cyber Security Division in the National Protection and Program Directorate.

The USAF Software Technology Support Center (STSC) is the publisher of CrossTalk providing both editorial oversight and technical review of the journal. CrossTalk's mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

Subscriptions: Visit <www.crosstalkonline.org/subscribe> to receive an e-mail notification when each new issue is published online or to subscribe to an RSS notification feed.

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CrossTalk editorial board prior to publication. Please follow the Author Guidelines, available at <www.crosstalkonline.org/submission-guidelines>. CrossTalk does not pay for submissions. Published articles remain the property of the authors and may be submitted to other publications. Security agency releases, clearances, and public affairs office approvals are the sole responsibility of the authors and their organizations.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CrossTalk.

Trademarks and Endorsements: CrossTalk is an authorized publication for members of the DoD. Contents of CrossTalk are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

CrossTalk Online Services:
For questions or concerns about crosstalkonline.org web content or functionality contact the CrossTalk webmaster at 801-417-3000 or webmaster@luminpublishing.com.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

CrossTalk is published six times a year by the U.S. Air Force STSC in concert with Lumin Publishing <luminpublishing.com>. ISSN 2160-1577 (print); ISSN 2160-1593 (online)

CrossTalk would like to thank 309 SMXG for sponsoring this issue.

Since the beginning of early civilization, humanity has established goals and succeeded in completing remarkable engineering accomplishments that pushed the boundaries of what was considered possible. One needs to look no further than the Great Pyramid of Giza (2584–2561 BC), Lighthouse of Alexandria, the Parthenon (447-438 BC) or Chichen Itza (c900 AD), for notable ancient examples. In recent times, we have gone from creating the monuments of ancient times to engineering complex architectures, structures, electronic systems and artificial intelligence, such as multinational undersea tunnels, artificial archipelagos, the International Space Station, and IBM's "Watson" - the new Jeopardy champion.

Major projects are not new, but the way in which we now manage projects has evolved.

The field of project management has continually transformed to address new challenges, primarily an ever-increasing growth in complexity and scope. Innovation has led to new ways of managing interrelationships between specialists performing a vast number of different tasks. Hardware and software are now coordinated in concert within a system design. Systems of systems deal with our increasing need for instantaneous information. Compartmentalization of core best practices for project management activities include integrated planning, organizing, resourcing, directing, monitoring, and issues resolution allow for greater reach and control over simultaneous and multifunctional tasks.

Software system development has brought its own unique challenges to the table, such as its intangible nature, clear requirements, feature creep, interoperability, defect detection, backwards compatibility, and the constant evolution of technology. Despite these challenges, the body of knowledge for managing these projects has grown substantially over the last few decades with new innovations in our engineering models, methods, techniques, activities, and especially the automation of tools.

In this issue, we have taken not only a historical perspective of past practices but also highlight new innovations that continue the advancement of project management as a discipline in its own right. We begin this issue with a collaborate work based upon a study conducted at The MITRE Corporation. This article highlights the benefits of including explicit software quality requirements at the proposal stage of government contract bids, which in turn would allow for contractor selection to be influenced by the use of best practices in software development. We continue our focus on quality with a fascinating analysis by Paul Croll in *Quality Attributes: Architecting Systems to Meet Customer Expectations*. This article emphasizes the importance of defining and using a set of quantifiable quality attributes tied to customer expectations when evaluating candidate system architectures. With a greater understanding of the relationship between quality attributes and architecture, we can better predict how candidate architectures will meet customer expectations.

Another pressing issue in software project management is the increasing complexity of projects and the inherent difficulties in managing emergent behavior in software systems. In *The Whole Is More Than the Sum of Its Parts: Understanding and Managing Emergent Behavior in Complex Systems*, the authors provide an overview of the increasing importance of applying systems theory to software as well as explore some speculative new methodologies for managing undesirable emergent behavior in complex systems. To illustrate some of the complexities we now face in software development, we now turn to *Developing a Model for Simplified Higher Level Sensor Fusion*. In this article, the authors systematically study the current difficulties faced by multisensory data fusion programs and ultimately provide an adaptation of models that can be used to provide an improved assessment while simplifying the process needed to get there.

In past issues, we have featured many articles that provide practical guidance to improve the quality of Earned Value Management (EVM) information and highlighted the value of such data in managing a project. In *Basing Earned Value on Technical Performance*, Paul Solomon readdresses the topic and proposes new solutions to further enhance the value of EVM. Continuing down the path of accurate and reliable information, we need to look no further than William Roetzheim's work in *Core Estimating Concepts*. This article reveals that beneath the myriad of domain-specific estimation tools available lies a set of core estimation concepts that can provide a framework for building new models for your specific needs. We conclude our set of articles by stressing the importance of peer reviews in *Statistical Tune-Up of the Peer Review Engine to Reduce Escapes*. In this article, Tom Lienhard identifies defects passing undetected through peer reviews as a major source of rework as a major problem and proposes innovative improvements to the peer review process. As always, be sure not to miss David Cook's humorous, yet insightful, look back at hard-learned lessons to writing good software.

As we begin the new year, we are also beginning the 25th year of CrossTalk publication as well. I would like to take a moment to express my sincere thanks to everyone for making such an accomplishment possible. To our co-sponsors, we thank you for your generous support and active involvement in providing an information and educational resource to the software industry. To the authors, we truly appreciate all of your time and effort in sharing such valuable information to the software community. To our readers, thank you for your continued support and hope that we continue to exceed expectations by publishing the highest quality articles.

From all of us at CrossTalk, we wish you the best for the new year!

Justin T. Hill
Publisher, CrossTalk

Applying the Fundamentals of Quality to Software Acquisition

Steve Bygren, The MITRE Corporation
Greg Carrier, The MITRE Corporation
Tom Maher, The MITRE Corporation
Patrick Maurer, The MITRE Corporation
David Smiley, The MITRE Corporation
Rick Spiewak, The MITRE Corporation
Christine Sweed, The MITRE Corporation

Abstract. Historically, software developed under government contracts often does not stand up under real-world use, and defects frequently result in cost and schedule overruns. While proposed development activities from contractors commonly list measures to improve quality, these descriptions cannot be used to select a winning bidder if they are not part of the evaluation criteria. By making software quality requirements explicit at the proposal stage, contractor selection can be influenced by criteria based on best practices in software development.

If we want to improve the quality of our software, a "Quality in Depth" approach is needed—introducing quality related measures at every stage of software acquisition. In a previous article,[1] one of the authors provided recommendations for improving software quality at the construction phase. This article discusses how to apply these same principles to the source selection process.

In order to find a way to include software practices as selection criteria, the authors set out to identify and recommend changes to Sections L and M of a government Request for Proposal (RFP) or Instructions for Proposal Preparation (IFPP) and Evaluation Criteria (EC) in an attempt to improve software and system quality. These changes will enable selection teams to identify contractors whose software development processes and compliance with software quality standards are more likely to produce the desired results.

1. Background

What Is Software Quality?

Quality is often thought of as an absence of defects. With many software products however, "defect" does not adequately describe the range of phenomena that affect software quality as perceived by the customers, end users and other stakeholders. Using Crosby's philosophy,[2] we define the term "software quality" to mean conformance to the requirements of the software product's users and other stakeholders. The more closely a software product conforms to these requirements, the higher its quality.

We are particularly interested in software quality as it affects the acquisition process for defense related software. While end user requirements are of prime importance, poor software development and quality monitoring practices in early- and mid-stage acquisition can result in failure to provide the desired results. These failures range from unwanted or missing features to cost and schedule overruns to critical flaws in system security or reliability.

How Do You Measure Software Quality?

Software quality as an outcome is best measured by the number of defects encountered after development is complete as the numerator, divided by the "size" of the software as the denominator. One could also argue that if two different products were to be compared, some sort of "difficulty factor" could be applied, as well as references to the software language or development environment employed, e.g., assembly code versus high order languages, or object-oriented versus functional languages, etc.

Metrics exist which can be used to estimate the potential defects in code. These are based on the use of function points as the measure of "size." Function points can also be (loosely) correlated with the commonly used measurement, SLOC.

2. Approach

This article is the outcome of a study the authors conducted at MITRE. Our approach was to gather information from Subject Matter Experts (SMEs), contracting officers, and acquisition experts for recommendations for additions to proposal documents. Part of this study was conducted through interviews and SME e-mail group lists. Reference materials from the Air Force and Navy were found which provided recommendations from prior work [1, 2]. We then adapted the suggestions to Sections L and M to more thoroughly describe software quality related criteria for source selection. Some of these criteria are aimed at the technical evaluation team, while some can be used by cost evaluators and past performance evaluators as well as the technical team.

3. Recommendations for Section L (Instructions for Proposal Preparation)

1. The offeror's proposal shall include a proposed Software Development Plan (SDP) which describes their approach to software development, to include the tools, techniques and standards to be used for development, unit testing and component testing; integration tools and techniques (including configuration management) used to ensure the integrity of system builds; the number and type of reviews that are part of the development process; and the methods and tools used to manage defect reports and analysis, including root cause analysis as necessary. The proposed SDP will form the basis for a completed SDP to be available after contract award as a Contract Deliverable Requirements List (CDRL) item, subject to government review and approval.

2. The offeror shall describe their plan for effective code reuse in order to minimize the amount of new code to be developed. Reused code can come from any origin, including previous efforts by the offeror or as provided by the Government in the bidders' library.

3. The offeror shall provide a Basis of Estimate (BOE) describing the rationale for the proposed staffing. The detail of the BOE shall include labor hours for each labor category (e.g., system engineering staff versus software engineering staff) for the identified tasks in the Work Breakdown Structure as it relates to the Statement of Work (SOW).

4. The offeror shall describe the process for orientation and training for all project employees (e.g. certification and training in software best practices including information assurance and risk management).

5. The offeror shall describe related systems experience, including a description of previous experience developing software of the same nature, and a description of the extent to which personnel who contributed to these previous efforts will be supporting this effort.

6. The offeror shall describe proposed development practices. For example, if spiral/incremental development, they shall describe the number, duration, and scope of spirals, as well as how the use of your approach would result in improved product quality and user satisfaction over time.[3]

7. The offeror shall provide an Integrated Master Schedule (IMS) and accompanying narrative that describes all significant program activities that are aligned with the proposed program staffing profile. Include a timeline for completion of each activity identified in the proposed program. Provide details that clearly describe the purpose for and importance of key activities. Identify all critical path elements and key dependencies.

4. Recommendations for Section M (Evaluation Criteria)

The proposed SDP shall show a complete and comprehensive software development process, which incorporates best practices as well as standards such as IEEE 12207-2008. The contractor will be evaluated based on how their processes, as described in the SDP, incorporate the use of software best practices.

Evaluation criteria related to the SDP include the following:

- The number and type of peer reviews.
- The use of automated unit testing including test coverage requirements.
- The use of automated syntax analysis tools and adherence to the rules incorporated by them.[4]
- The comprehensiveness of integration and test methods, including continuous integration tools if used.
- The use of readiness requirements such as unit test and syntax analysis for code check-in.
- Configuration management and source code control tools and techniques.
- The extent to which root cause analysis of defects is part of the development process.
- The selection of software source code to be reused, replaced or rewritten from previous implementations or other origins, including a description of how it will be ensured that reused code meets or is brought up to the same standards as newly developed code. Risks associated with reused software shall also be discussed. Such software shall include government rights to the source code.

The IMS and accompanying narrative will be evaluated for level of detail and relevance of significant program activities, degree of alignment, the proposed program staffing profile, and integration of the proposed SDP into the IMS. Additionally, critical path elements and key dependencies will be assessed for relevance, completeness and the manner and level of risk containment.

Parameter/rating	Unacceptable	Marginal	Acceptable	Superior
The number and type of peer reviews	none	1 (any)	2 (design, code)	3 or more (requirements, design, code, test)
The use of automated unit testing including test coverage requirements	none	unit tests written after manual testing or only on selected code	automated tests 75% code coverage on new or modified code	automated tests 85% or more code coverage on all delivered code. The use of Test Driven Development.
The use of automated syntax analysis tools and adherence to the rules incorporated by them	none	used selectively or with heavily modified rules	used consistently with standard rules	additional rules or tools specific to security analysis
The comprehensiveness of integration and test methods including continuous integration tools if used	ad-hoc	formal integration and test	automated processes applied periodically	continuous integration including syntax analysis and unit tests
The use of readiness requirements such as unit test and syntax analysis for code check-in	none	individual manual testing	integrated testing by developer	automated part of check-in and continuous integration process
Configuration management and source code control tools and techniques	manual/paper trail	by individual developer	system-wide repository	managed tool with pre-check-in requirements
The extent to which root cause analysis of defects is part of the development process	none	"red-team" only	serious defects	routine periodic analysis of defect pool
The selection of software source code to be reused, replaced, or re-written from previous implementations	none or no response	replacement with contractor's previous work	rework of selected items showing good knowledge of base software	innovative approach to maximum reuse and modernization

Table 1. Sample Rating Scale for SDP Evaluation Criteria[5]

5. Incorporating Software Quality Measures in Contracts

The contract development process includes several steps at which information can be gathered and requirements set to include software quality as a measure of vendor performance.

Sections L & M or equivalent from the RFP

>> Add software quality measures as a discriminating factor in selecting the contractor

>> Enumerate expectations in this area:
- Types of methods used
- Evidence to be provided

Technical Requirements Document, Statement of Objectives, and SOW

Add requirements in the form of deliverable items—as CDRLs or Data Accession List items as appropriate. Examples include the following:

>> Output of automated unit tests showing code coverage at or above required minimum.

>> Output of automated syntax analysis showing conformance to pre-determined rules.

>> Evidence of accomplishing required peer reviews.

>> Itemized list of tools with version numbers used to produce output from each source module.

>> Programmer's reference manual with examples.

>> Interface definitions.
>> List of all software components with the following information:
  • Purpose and function.
  • Interfaces provided.
  • Language/version for each module.
  • Complete source code.
>> Source from architectural design tool where available.
>> Use cases (text and diagrams).
>> Class diagrams where applicable.
>> Complete list of any third-party components with version numbers.
>> Contact information for any outside dependencies.
>> Build procedures, including documentation for building all software components from source code.
>> Test procedures—including any automated unit tests with source code, test scripts.

6. Rationale for Incorporating Recommended RFP Language

The recommended RFP language was derived by the authors from a variety of sources including MITRE acquisition subject matter experts, existing guidance documents from the Navy and Air Force, and also from the authors' experience. We have tried to provide a succinct rationale as to why the language asks for specific information from the contractor in the RFP:

The SDP is a maturity indicator of the bidder's development process. By evaluating this, and then putting its provisions under contract, it becomes possible to select a contractor on the basis of development methodology and then obligate them to perform as proposed.

Automated unit tests and comprehensive peer reviews are widely used best practices. Capers Jones[6] has noted that these are among the required steps to achieve effective defect removal.

Continuous Integration (CI) often includes the automated invocation of tests and code analysis during the build process. CI and static analysis expose problems earlier in the development process. The earlier problems are discovered, the lower the cost to resolve.

Root cause analysis prevents the introduction of defects and is a recognized best practice in all approaches to process improvement. It is a CMMI® Level 5 practice area. Prevention is more cost effective than detecting and fixing defects after they are introduced.

The BOE helps the evaluator understand the bidder's cost to compare against industry averages and government cost models. By examining proposed labor categories, this can be checked against predicted labor distributions from government cost models as well.

The IMS can be checked for alignment with required milestone dates, and it supports an independent estimate.

7. Guidance for Evaluation Team Experience

The government's evaluation team must have relevant software engineering experience. The experience should cover the full life cycle of software development from design to development, integration, testing, and delivery. If the proposal is seeking a particular style of development methodology (e.g., waterfall, spiral/incremental, agile), then the evaluation team should have experience in that methodology in order to evaluate the RFP response.

Since a significant portion of the suggested contract language relates to software quality monitoring, the evaluators should be familiar with unit testing, peer reviews, CI, static code analysis, and metrics. Finally, evaluators should have some knowledge of various practices and approaches of applying these techniques, for example, when it comes to test-driven development.

The field of software engineering is diverse. It is insufficient to simply have general software engineering experience on the evaluation team without further having experience in the applicable domain(s). Examples of these domains include real-time/embedded, kernel/operating systems, numerical/digital signal processing, web applications, SOA, information retrieval/search, security, and human-computer interface.

Finally, the evaluation team should have an understanding of the CMMI process and rating criteria.

8. Guidance for Evaluating Technical Responses

The recommended contract language in this article includes Section M of the RFP, also appearing as Evaluation Criteria. The language is not very specific so as to elicit responses that are more original than simply claiming to do a long list of things that the government is checking for. In this section, we discuss more specific guidance for the evaluation team in evaluating the responses.

In advance, the team should define objectives that are sought after and then define measurable criteria. The more objective the criteria, the better, though it is recognized that coming up with this criteria can be a challenge. After defining criteria, they are prioritized and then weighted in a scheme the team deems appropriate.

Some general evaluation tips are as follows:

• If key staff are identified in the proposal, how likely are they to be available during contract execution?

• In reference to quality assurance processes, does the proposal language favor or at least mention "empowerment" of the quality assurance team over engineering processes?

• Regarding the contractor's approach to automated unit testing: Does the contractor require that unit tests be passed and cover a reasonable percentage of code before code can be checked in? Does the contractor use test-driven development?

• Regarding the contractor's approach to automated syntax analysis: Does the contractor require that syntax analysis be performed and that all required rules are followed before code can be checked in?

• Regarding development build and integration: Does the contractor use an automated build process that incorporates syntax analysis and automated unit testing?

You can expect that the response is going to claim appraisal at a specific CMMI maturity level (commonly at least level 3). This can be verified with the Appraisal Disclosure Statement (ADS) document. Another source is the Standard CMMI Appraisal Method for Process Improvement (SCAMPI). For the larger contractors, particularly when work is further sub-contracted out, look for further CMMI level compliance information on the specific division/unit and sub-contractor(s) as applicable.

9. Development Process

If the proposal declares that a development process will be used that will involve multiple iterations/spirals/increments (which is standard practice), then the evaluation team should look for further details on the process to include the following:

• What is the duration and scope of each increment?

• Are lessons and obstacles from one increment reviewed for improvement to a subsequent increment?

• Is user (customer) feedback interaction only up front or do most increments incorporate this? And how is that feedback prioritized?

• Are multiple increments planned in sufficient detail, or are only the present and possibly next increment planned?

10. Software Engineering

One key thing to look for in a proposal is to what degree the contractor has experience in the technology the RFP calls for them to deliver. The more complex the system, the more important applicable contractor experience is.

Many DoD systems have a degree of interoperability and integration required of them. For integration with particular systems, verify if the contractor has experience with that system or has relationships with third parties with integration capabilities that will be used. The contractor should also participate in applicable Communities of Interest.

Testing processes and technologies that support them are important. Look for information on a test plan or strategy. If the proposal is serious about continuous integration and use of supporting tools, then listing the software to be used for this is a promising sign. Information on how the tools are used (e.g., by exception and/or monitored on a periodic basis—and what period) is also telling. If the proposal includes information on the proposed system design, then the evaluators could look to see how "testable" the design is, particularly as it is incrementally built.

11. Conclusions

While it is important to implement quality measures in software construction, this is undertaken after a contractor has been selected. The authors recommend an in-depth approach, beginning with the process of selecting the contractor. It can be easy to overlook the importance of including specific language in the proposal documents in order to be able to select the right contractor from those responding to an RFP. In order to accomplish this goal, it is critical to specify the instructions in Section L (or the IFPP) and the evaluation criteria in Section M (or the EC) so that these can be used to assign strengths or weaknesses appropriately. This is an early, but often neglected, piece of the puzzle involved in building quality software products for defense applications.

Acknowledgement:

The authors wish to thank Tim Aiken and Carole Mahoney for their guidance as we conducted our research. Additionally, we wish to thank the MITRE Systems Engineering Practice Office, the acquisition community, and our various programs for their contributions to our survey.

Disclaimers:

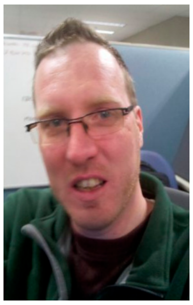
ABOUT THE AUTHORS

Steve Bygren is a Principal Information Systems Engineer at The MITRE Corporation, supporting the Electronics System Center (ESC) from Peterson Air Force Base, Colorado, where his focus is on multi-system integration and advanced development. Bygren has 28 years of experience in software and systems engineering, and received his Bachelor of Science in Computer Science from Montana State University.

The MITRE Corporation
1155 Academy Park Loop
Colorado Springs, CO 80910
Phone: 719-659-3794
E-mail: sbygren@mitre.org

Greg Carrier is a lead software systems engineer at the MITRE Corporation in Bedford, MA. Greg's interest in improving software quality stems from his experience managing software-related projects for a variety of government contracts.

The MITRE Corporation
202 Burlington Rd.
MS S355
Bedford, MA 01730
Phone: 781-271-5180
E-mail: gcarrier@mitre.org



Tom Maher is a Senior Software Systems Engineer for the Information and Computing Technologies Division at MITRE with more than 15 years of hands-on experience in applying technology to meet the needs of business. Tom's employers and clients have ranged from Internet startups to divisions of multi-billion dollar corporations; he has worked across multiple domains including retail, financial services, medical information systems, and homeland defense.

The MITRE Corporation
202 Burlington Rd.
MS 1630T
Bedford, MA 01730
Phone: 781-225-5355
E-mail: tdmaher@mitre.org

ABOUT THE AUTHORS

Patrick Maurer is a lead communications engineer at The MITRE Corporation. He develops networking and network management technologies for SATCOM and line-of-sight terminals. His current work focuses on standardizing network management interface for terminals. Prior to joining MITRE, Patrick worked in the telecommunications industry developing modems and networking systems. He has a BSEE from Northeastern University, MSEE from Massachusetts Institute of Technology and an MBA from Boston University.

The MITRE Corporation
MS D300
202 Burlington Road
Bedford, MA 01730
Phone: 781-271-4698
E-mail: pmaurer@mitre.org

David Smiley is a lead software developer specializing in search technologies. He has 12 years of experience in the defense industry at MITRE using Java and various web technologies. David is the principal author of "Apache Solr 3 Enterprise Search Server" and has presented at conferences and taught classes about Solr.

The MITRE Corporation
202 Burlington Rd.
MS M330
Bedford, MA 01730
Phone: 781-271-7659
E-mail: dsmiley@mitre.org

Rick Spiewak is a Lead Software Systems Engineer at The MITRE Corporation. Rick works at the Electronic Systems Center at Hanscom Air Force Base as part of the Battle Management Group, concentrating on Mission Planning. He has been focusing on the software quality improvement process, and has spoken on this topic at several conferences. Rick has been in the computer software industry for more than 41 years, and has bachelor's and master's degrees in Electrical Engineering from Cornell University. He studied quality management at Philip Crosby Associates.

The MITRE Corporation
202 Burlington Rd.
MS 1614E
Bedford, MA 01730
Phone: 781-225-9298
E-mail: rspiewak@mitre.org

Christine Sweed is a Lead Networking Systems and Distributed Systems Engineer at The MITRE Corporation. She has worked at MITRE for more than 10 years on various software development projects following 10 years of industry experience. She has a B.A. in chemistry from SUNY Potsdam, and a M.S. in Computer Science from Boston University

The MITRE Corporation
202 Burlington Rd.
MS K214
Bedford, MA 01730
Phone: 781-271-3552
E-mail: csweed@mitre.org

REFERENCES

1. USAF Weapon Systems Software Management Guide, August 2008. <https://acc.dau.mil/adl/en-US/24374/file/49721/USAF%20WSSMG%20%20ABRIDGED.pdf>.
2. Guidebook for Acquisition of Naval Software Intensive Systems, dated September 2008. <https://acquisition.navy.mil/rda/content/download/5657/25845/version/1/file/Guidebook+for+Acquisition+of+Naval+Software+Intensive+SystemsSEP08.pdf>.

NOTES

1. Spiewak, Rick and Karen McRitchie. "Apply the Fundamentals of Quality in Software Construction to Reduce Costs and Prevent Defects." CrossTalk, Dec. 2008. <http://www.crosstalkonline.org/storage/issue-archives/2008/200812/200812-Spiewak.pdf>
2. Crosby, Philip B. Quality Is Free: The Art of Making Quality Certain. McGraw-Hill Companies, 1979.
3. Note that while not part of the technical evaluation, the government evaluation team will examine Contractor Performance Assessment Reports (CPARs) for relevant performance by the respondent on other contracts.
4. Jones, Capers. Software Engineering Best Practices. McGraw-Hill, 2010
5. Categories suggested by conversation with Jeff Pattee, Chief, Product Definition, Airspace Mission Planning Division, Electronic Systems Center, USAF.
6. Jones, Capers. "Measuring Defect Potentials and Defect Removal Efficiency." CrossTalk, June 2008. <http://www.crosstalkonline.org/storage/issue-archives/2008/200806/200806-Jones.pdf>.

Quality Attributes

Architecting Systems to Meet Customer Expectations

Paul R. Croll, CSC

Abstract. This paper addresses the use of quality attributes as a mechanism for making objective decisions about architectural tradeoffs and for providing reasonably accurate predictions about how well candidate architectures will meet customer expectations. Typical quality attributes important to many current systems of interest include performance, dependability, security, and safety. This paper begins with an examination of how quality attributes and architectures are related, including some the seminal work in the area, and a survey of the current standards addressing product quality and evaluation. The implications for both the customer and the system developer of employing a quality-attribute-based approach to architecture definition and tradeoff are then briefly explored. The paper also touches on the relationship of an architectural quality-attribute-based approach to engineering process and process maturity. Lastly the special concerns of architecting for system assurance are addressed.

It Is About the Architecture

A recent presentation on systemic root cause analysis of failures in DoD programs [1] pointed out that:

"DoD operational test and evaluation results from October 2001 through September 2006 indicated that of 29 systems evaluated, approximately. 50% were deemed 'Not Suitable', or 'partially Not Suitable' and approximately 33% were deemed 'Not Effective', or 'partially Not Effective'."

The presentation went on to say that one of the top 10 emerging systemic issues, from 52 in-depth program reviews since March 2004 was inadequate software architectures.

If we are to be successful in delivering systems that meet customer expectations, we must start as early as possible in the design process to understand the extent to which those expectations might be achieved. As we develop candidate system architectures and perform our architecture tradeoffs, it is imperative that we define and use a set of quantifiable system attributes tied to customer expectations, against which we can measure success.

In 2006, the National Defense Industrial Association (NDIA) convened a Top Software Issues Workshop [2] to examine the current most critical issues in software engineering that impact the acquisition and successful deployment of software-intensive systems.

The workshop identified 85 issues for further discussion, which were consolidated into a list of the top seven. Of those issues impacting software-intensive systems throughout the lifecycle, two emerged that were focused specifically on the relationship between software quality and architecture:

- Ensure defined quality attributes are addressed in requirements, architecture, and design.
- Define software assurance quality attributes that can be addressed during architectural tradeoffs.

As is true in the defense systems case above, most systems we encounter today contain software elements and most depend upon those software elements for a good portion of their functionality. Modern systems architecture issues cannot be adequately addressed without considering the implications of software architecture.

Architecture and Quality

What is an architecture? IEEE Std 1471-2000 [3, 34] defines an architecture for software intensive systems as:

"The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution."

More recently, Firesmith et al [4], their Method Framework for Engineering System Architectures (MFESA), have defined system architecture as:

"The set of all of the most important, pervasive, higher-level, strategic decisions, inventions, engineering tradeoffs, assumptions, and their associated rationales concerning how the system meets its allocated and derived product and process requirements."

The authors believe that system architecture is a major determinant of resulting system quality.

MFESA instructs that architectures can be represented by models, views, and focus areas. Models describe system structures in terms of their architectural elements and the relationships between them. These descriptions can be graphical or textual and include the familiar data and control flow diagrams, entity-relationship diagrams, and UML diagrams and associated use cases. Views are composed of one or more related architectural models. They use the example of a class view that describes all architectural classes and their relationships. Focus areas combine multiple views and models to determine how the architecture achieves specific quality characteristics.

What is quality and what are quality characteristics? IEEE Standard 1061-1998 [5], defines software quality as the degree to which software possesses a desired combination of attributes.

Similarly, ISO/IEC 9126-1:2001 [6], one of a four-part set of standards on software product quality, defines quality as:

"The totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs."

The standard identifies a quality model with six quality characteristics: functionality, reliability, usability, efficiency, maintainability and portability. The other three standards in the 9126-series [7, 8, 9] address metrics for measuring attributes of the quality characteristics defined in ISO/IEC 9126-1.

It should be noted that the 9126-series is being revised as part of the Software Product Quality Requirements and Evaluation (SQuaRE) series of standards. ISO/IEC 25010, Software engineering—SQuaRE—quality model [10] is the revision of ISO/IEC 9126-1:2001. ISO/IEC 25010 adds security and interoperability to the list of six quality characteristics defined in ISO/IEC 9126-1:2001. Additionally ISO/IEC 25030, Software engineering—SQuaRE)—quality requirements [11] defines the concept of internal software quality as the "capability of a set of static attributes (including those related to software architecture) to satisfy stated and implied needs when the software product is used under specified conditions." and the concept of software quality in use, which is "the capability of the software product to enable specific users to achieve specific goals with effectiveness, productivity, safety and satisfaction in specific contexts of use."

Functional properties determine what the software is able to do. Quality properties determine how well the software performs. In other words, the quality properties show the degree to which the software is able to provide and maintain its specified services.

ISO/IEC/IEEE 15288 [12] addresses the confluence of architecture and quality in the context of the system lifecycle. The Architectural Design Process (6.4.3) provides for the creation of design criteria for quality characteristics and the evaluation of alternative designs with respect to those criteria. There is also a Specialty Engineering view of the lifecycle processes in that focuses on the achievement of product characteristics that have been selected as being of special interest.

Quality Attribute-based Approaches to Architecting Systems

In the seminal report on Quality Attributes by Barbacci et al [13] the authors indicate that:

"Developing systematic ways to relate the software quality attributes of a system to the system's architecture provides a sound basis for making objective decisions about design tradeoffs and enables engineers to make reasonably accurate predictions about a system's attributes that are free from bias and hidden assumptions. The ultimate goal is the ability to quantitatively evaluate and trade off multiple software quality attributes to arrive at a better overall system."

Franch and Carvallo [14] suggest that for an effective quality model, the relationships between quality attributes must be explicitly stated to understand potential attribute clash when defining software architectures. They posit three types of relationships between attributes:

- Collaboration, in which increasing the degree to which one attribute is realized increases the realization of another.
- Damage, in which increasing the degree to which one attribute is realized decreases the realization of another.
- Dependency, in which the degree to which one attribute is realized, is dependent upon the realization of at least some sub-characteristics of another.

For example, as Häggander et al [15] point out using the example of a large telecommunication application, system architects must balance multiple quality attributes, such as maintainability, performance and availability. Focusing solely on the attribute of maintainability often results in poor system performance and conversely focusing on performance and availability alone may result in result in poor maintainability. Explicit architectural decisions can facilitate optimization among quality attributes.

Architectural Design and Tradeoff

Bass and Kazman [16] suggest five foundational structures that together completely describe an architecture and that can serve as the basis for understanding the relationship of architectural decisions to quality attributes:

- Functional structure is the decomposition of the functionality that the system needs to support
- Code structure is the code abstractions from which the system is built.
- Concurrency structure is the representation of logical con currency among the components of the system.
- Physical structure is just that, the structure of the physical components of the system.
- Developmental structure is the structure of the files and the directories identifying the system configuration as the system evolves.

Bass and Kazman [16] further suggest some likely relationships between the architectural structures described above and examination of the impact of architectural decisions upon specific quality attributes. They suggest for example that:

- Concurrency and physical structures are useful in understanding system Performance.
- Concurrency and code structures are useful in understanding system security.
- Functional, code, and developmental structures are useful in understanding system maintainability.

Wojcik et al [17] describe an Attribute-driven Design (ADD) method in which the approach to defining software architecture is based on software quality attribute requirements. ADD produces an initial software architecture description from a set of design decisions that show:

- Partitioning of the system into major computational and developmental elements.
- What elements will be part of the different system structures, their type, and the properties and structural relations they possess.
- What interactions will occur among elements, the properties of those interactions, and the mechanisms by which they occur.

In the very first step in ADD, quality attributes are expressed as the system's desired measurable quality attribute response to a specific stimulus. Knowing these requirements for each quality attribute supports the selection of design patterns and tactics to achieve those requirements.

Kazman et al [18] describe an Architecture Tradeoff Analysis Method (ATAM) that can be used when evaluating an architecture, including those produced by the ADD method above, in order to understand the consequences of architectural decisions with respect to quality attributes. As the authors point out, ATAM is dependent upon quality attribute characterizations, like those produced through ADD, that provide the following information about each attribute:

- The stimuli to which the architecture must respond.
- How the quality attribute will be measured or observed to determine how well it has been achieved.
- The key architectural decisions that impact achieving the attribute requirement.

WANTED

Electrical Engineers and Computer Scientists
Be on the Cutting Edge of Software Development

The Software Maintenance Group at Hill Air Force Base is recruiting **civilians** *(U.S. Citizenship Required)*. Benefits include paid vacation, health care plans, matching retirement fund, tuition assistance, and time paid for fitness activities. **Become part of the best and brightest!**

Hill Air Force Base is located close to the Wasatch and Uinta mountains with many recreational opportunities available.

facebook
www.facebook.com/309SoftwareMaintenanceGroup

Send resumes to:
309SMXG.SODO@hill.af.mil
or call (801) 775-5555

ATAM takes proposed architectural approaches and analyzes them based upon quality attributes, generally specified in terms of scenarios addressing stimuli and responses. ATAM also identifies sensitivity points and tradeoff points.

ATAM describes stakeholders' interaction with the system. Stakeholders bring different views to the system and may include users, maintainers, developers, and acquirers. Scenarios specify the kinds of operations over which performance needs to be measured, or the kinds of failures the system will have to withstand. ATAM uses three types of scenarios:

- Use case scenarios, describing typical uses of the system.
- Growth scenarios, addressing planned changes to the system.
- Exploratory scenarios, addressing any possible extreme changes that would stress the system.

Making the Case for Architectural Quality

How do stakeholders know that the system will exhibit expected quality characteristics? Firesmith et al [4] suggest that one method is the quality case, or more specifically for evaluating architectures, the architectural quality case. Quality cases consist of the set of claims, supporting arguments, and supporting evidence that provide confidence that the system will in fact demonstrate its expected quality characteristics. Common types of quality cases include safety cases [19], and security cases [20], and the more generalized assurance cases [21]. Architectural quality cases describe the architectural claims, supporting arguments, including architectural decisions and tradeoffs, architectural representations, and demonstrations that the architecture will exhibit its expected quality characteristics.

The implications for both the customer and the system developer of employing a quality-attribute- based approach to architecture definition and tradeoff, documented in part by a quality case, are that:

- Customer quality requirements will have been distilled into architectural drivers [17] that will have shaped the system architecture.
- Tradeoffs will have been made to optimize the realization of important quality characteristics, in concert with customer expectations.
- The level of confidence that the resultant architecture will meet those expectations will be known.
- Customers will be knowledgeable of any residual risk they are accepting by accepting the delivered system.

There are also architectural implications regarding sustainment of a system over its lifecycle. Croll [22] cites that with respect to sustainment, paying insufficient attention to sustainment issues early in the lifecycle, including licensing, and product support can lead to problems when commercial products or components inevitably change or when their suppliers either discontinue support or go out of business. In the hardware world Diminishing Manufacturing Sources and Materials Shortage (DMSMS) analyses are generally done when integrating commercial components as part of an approach for managing the risk of obsolescence [23]. DMSMS analyses focus on supplier viability for the product of interest, which could be considered an attribute of component maintainability. Certainly, such analyses, where necessary, should be part of the quality case. The specification and realization of architectures which are resilient with respect to the substitution of alternate software components can further enhance system quality through the lifecycle.

Process Maturity Does Not Guarantee Product Quality

We spend much time these days focusing on the maturity of our engineering processes and heralding process maturity ratings such as those associated with the CMMI® [24], for development and the ISO 9000 series [25][26][27], for quality management systems, as indicators of our ability to deliver quality products – products that meet the customer's expectations and that continue to do so throughout their lifecycle. What our customers have found, however, is that often process maturity does not guarantee product quality. This is especially true for the highly software intensive systems we now build, where performance, dependability, and failure modes are less well understood.

For example, although the CMMI embodies the process management premise that, the quality of a system or product is highly influenced by the quality of the process used to develop and maintain it [24], Hefner [28] points out:

Several recent program failures from organizations claiming high maturity levels have caused some to doubt whether CMMI improves the chances of a successful project.

He goes on to say, "an CMMI appraisal indicates the organization's capacity to perform the next project, but cannot guarantee that each new project will perform in that way."

Understanding and Leveraging a Supplier's CMMI Efforts: A Guidebook for Acquirers [29] further underscores the problem and offers several cautions for acquirers, with respect to supplier claims of process maturity.

- A CMMI rating or CMMI level is not a guarantee of program success.
- Organizations that have attained CMMI maturity level ratings do not necessarily apply those appraised processes to a new program at program startup.
- Organizations that claim CMMI ratings are not always dedicated to [maintaining] process improvement [through out the development effort].
- Organizations may sample only a few exemplar programs and declare that all programs are being executed at that CMMI level rating.
- Organizations that claim a high maturity level rating (level 4 and 5) are not necessarily better suppliers than a level 3 supplier. Maturity levels 4 and 5, when compared across different suppliers, are not created equal.

Although process maturity can in many cases improve project performance [30], special attention to the engineering processes is required to ensure that customer quality expectations are realized in resultant products.

A Current Concern: Architecting for System Assurance

Stakeholder discussion over the last several years has demonstrated a reasonably consistent view of the problem space. System assurance can be viewed as the level of confidence that the system functions as intended and is free of exploitable vulnerabilities, either intentionally or unintentionally designed or inserted as part of the system. The President's Information Technology Advisory Committee report entitled Cyber Security: A Crisis of Prioritization [31] states, "… the approach of patching and retrofitting networks, computing systems, and software to 'add' security and reliability may be necessary in the short run but is inadequate for addressing the Nation's cyber security needs." The report further suggests, "we simply do not know how to model, design, and build systems incorporating integral security attributes."

As Croll points out [22], the systems engineering challenge, with respect to assurance, is in integrating a heterogeneous set of globally engineered and supplied proprietary, open-source, and other software; hardware; and firmware; as well as legacy systems; to create well-engineered integrated, interoperable, and extendable systems whose security, safety, and other risks are acceptable—or at least tolerable.

Baldwin [32] underscores this challenge for DoD systems by describing a vision for assurance in which the requirements for assurance are allocated among the right systems and their critical components, and such systems are designed and sustained at a known level of assurance.

The National Defense Industrial Association System Assurance Guidebook [33] describes practices in architectural design that can improve assurance. The Guidebook suggests some general architectural principles for assurance:

- Isolate critical components from less-critical components.
- Make critical components easier to assure by making them smaller and less complex.
- Separate data and limit data and control flows.
- Include defensive components whose job is to protect other components from each other and/or the surrounding environment.
- Beware of maximizing performance to the detriment of assurance.

The Guidebook also suggests using system assurance requirements, design constraints and system assurance critical scenarios for architectural tradeoff analysis, and documenting the results in the assurance case.

Summary

If we are to be successful in delivering systems that meet customer expectations, we must start as early as possible in the design process to understand the extent to which those expectations might be achieved. As we develop candidate system architectures and perform our architecture tradeoffs, it is imperative that we define and use a set of quantifiable quality attributes tied to customer expectations, against which we can measure success.

Standards like ISO/IEC TR 9126, Parts 1-4, ISO/IEC 25010, and ISO/IEC 2530 can help stakeholders define quality attributes from both an internal perspective, useful for addressing architectural design, and a quality in use perspective addressing system realization.

Methods have been documented to aid in understanding the relationship of architectural decisions to quality attributes, for defining software architecture is based on software quality attribute requirements, and for understanding the consequences of architectural decisions with respect to quality attributes.

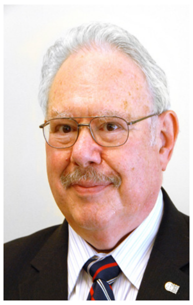
Architectural quality cases describe the architectural claims, supporting arguments, including architectural decisions and tradeoffs, architectural representations, and demonstrations that the architecture will exhibit its expected quality characteristics. They are extremely useful in providing customers with an understanding of any residual risk they are accepting by accepting the delivered system.

Several recent program failures from organizations claiming high maturity levels have caused some doubt about whether process maturity improves the chances of a delivering a successful product. This is especially true for the highly software intensive systems we now build, where performance, dependability, and failure modes are less well understood.

Of special concern these days is architecting systems for system assurance. Given our track record in architecting systems to meet assurance concerns, guidance is needed to support assurance-specific architectural design and tradeoff analysis, as well as appropriate documentation of assurance claims, arguments, and supporting evidence, so that customers understand the degree to which the architecture mitigates assurance risks.

Disclaimers:

ABOUT THE AUTHOR



Paul Croll is a Fellow in CSC's Defense Group where he is responsible for researching, developing and deploying systems and software engineering practices, including practices for cybersecurity.

Paul has more than 35 years experience in mission-critical systems and software engineering. His experience spans the full lifecycle and includes requirements specification, architecture, design, development, verification, validation, test and evaluation, and sustainment for complex systems and systems-of-systems. He has brought his skills to high profile, cutting edge technology programs in areas as diverse as surface warfare, air traffic control, computerized adaptive testing, and nuclear power generation.

Paul is also the IEEE Computer Society Vice President for Technical and Conference Activities, and has been an active Computer Society volunteer for more than 25 years, working primarily to engage researchers, educators, and practitioners in advancing the state of the practice in software and systems engineering. He was most recently Chair of the Technical Council on Software Engineering and is also the current Chair of the IEEE Software and Systems Engineering Standards Committee. Paul is also the past Chair and current Vice Chair of the ISO/IEC JTC1/SC7 U.S. Technical Advisory Group (SC7 TAG).

Paul is also active in industry organizations and is the Chair of the NDIA Software Industry Experts Panel and the Industry Co-Chair for the National Defense Industrial Association Software and Systems Assurance Committees. In addition, Paul is Co-Chair of the DHS/DoD/NIST Software Assurance Forum Processes and Practices Working Group advancing cybersecurity awareness and practice.

CSC
17021 Combs Drive
King George, VA 22485
Phone: 540-644 6224
E-mail: pcroll@csc.com

REFERENCES

1. D. Castellano. Systemic Root Cause Analysis. NDIA Systems Engineering Division Strategic Planning Meeting, December, 2007.
2. G. Draper (ed.), Top Software Engineering Issues Within Department of Defense and Defense Industry. National Defense Industrial Association, Arlington, VA, August 2006.
3. IEEE 1471-2000, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. The Institute of Electrical and Electronics Engineers, Inc., New York, NY, 2000.
4. D. Firesmith, P. Capell, D. Falkenthal, C. Hammons, D. Latimer, and T. Merendino. The Method-Framework for Engineering System Architectures (MFESA): Generating Effective and Efficient Project-Specific System Architecture Engineering Methods. To be published.
5. IEEE Standard 1061-1992. Standard for a Software Quality Metrics Methodology. New York: Institute of Electrical and Electronics Engineers, 1992.
6. ISO/IEC 9126-1: Information Technology - Software product quality - Part 1: Quality model. ISO, Geneva Switzerland, 2001.
7. ISO/IEC TR 9126-2: Software Engineering - Product quality - Part 2: External metrics. ISO/IEC, Geneva Switzerland, 2003.
8. ISO/IEC TR 9126-3 Software engineering – Product quality - Part 3: Internal metrics. ISO/IEC, Geneva Switzerland, 2003.
9. ISO/IEC TR 9126-4: Software engineering – Product quality - Part 4: Quality in use metrics. ISO/IEC, Geneva Switzerland, 2004.
10. ISO/IEC CD 25010, Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) Quality model. ISO/IEC, Geneva, Switzerland, 2007.
11. ISO/IEC CD 25030, Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Quality requirements. ISO/IEC, Geneva Switzerland, 2007.
12. ISO/IEC 15288:2002, Systems Engineering - System Life Cycle Processes, ISO/IEC, Geneva Switzerland, 2002.
13. M. Barbacci, M. Klein, T. Longstaff, and C. Weinstock. Quality Attributes, CMU/SEI-95-TR-021. Software Engineering Institute, Carnegie Mellon University, December 1995.
14. X. Franch and J. Carvallo. "Using Quality Models in Software Package Selection", IEEE Software, pp. 34-41. New York: Institute of Electrical and Electronics Engineers, 2003.
15. D. Häggander, L. Lundberg, and J. Matton, "Quality Attribute Conflicts - Experiences from a Large Telecommunication Application," Proceedings of the Seventh International Conference on Engineering of Complex Computer Systems (ICECCS'01), New York: Institute of Electrical and Electronics Engineers, 2001.
16. L. Bass and R. Kazman, Architecture-Based Development, CMU/SEI-99-TR-007. Software Engineering Institute, Carnegie Mellon University, April 1999.
17. R. Wojcik, F. Bachmann, L. Bass, P. Clements, P. Merson, R. Nord, and B. Wood, Attribute-Driven Design (ADD), Version 2.0, CMU/SEI-2006-TR-023. Software Engineering Institute, Carnegie Mellon University, November 2006.
18. R. Kazman, M. Klein, and P. Clements, ATAM: Method for Architecture Evaluation, CMU/SEI-2000-TR-004, Software Engineering Institute, Carnegie Mellon University, August 2000.
19. W. Greenwell, E. Strunk, and J. Knight, Failure Analysis and the Safety-Case Lifecycle, IFIP Working Conference on Human Error, Safety and System Development (HESSD) Toulouse, France, August 2004.
20. Systems Security Engineering Capability Maturity Model®, SSE-CMM®, Model Description Document Version 3.0. Systems Security Engineering Capability Maturity Model (SSE- CMM) Project (Copyright © 1999), June 15, 2003
21. T. Ankrum and A. Kromholz, "Structured Assurance Cases: Three Common Standards, Proceedings of the Ninth IEEE International Symposium on High-Assurance Systems Engineering (HASE'05). New York: Institute of Electrical and Electronics Engineers, 2005.
22. P. Croll, "Engineering for System Assurance – A State of the Practice Report," Proceedings of the 1st Annual IEEE Systems Conference. New York: Institute of Electrical and Electronics Engineers, April 2007.
23. Diminishing Manufacturing Sources and Material Shortages (DMSMS) Guidebook. Office of the Under Secretary of Defense Acquisition, Technology, & Logistics, November 2006.
24. CMMI® for Development, Version 1.2, CMU/SEI-2006-TR-008, Software Engineering Institute, Carnegie Mellon University, August 2006.
25. ISO 9000:2000, Quality management systems – Fundamentals and vocabulary. ISO, Geneva Switzerland, 2000.
26. ISO 9001:2000, Quality management systems – Requirements. ISO, Geneva Switzerland, 2000.
27. ISO 9004:2000 Quality management systems – Guidelines for performance improvements. ISO, Geneva Switzerland, 2000.
28. R. Hefner. CMMI Horror Stories: When Good Projects Go Bad. SEPG Conference, March 2006
29. Understanding and Leveraging a Supplier's CMMI® Efforts: A Guidebook for Acquirers, CMU/SEI-2007-TR-004. Software Engineering Institute, Carnegie Mellon University, March 2007.
30. D. Goldenson and D. Gibson, Measuring Performance: Evidence about the Results of CMMI®. 5th Annual CMMI Technology Conference & User Group. National Defense Industrial Association, System Assurance Committee, Arlington, Virginia, November 2005.
31. President's Information Technology Advisory Committee (PITAC), Cyber Security: A Crisis of Prioritization. National Coordination Office for Information Technology, Arlington, VA, 2005.
32. K. Baldwin. DOD Software Engineering and System Assurance New Organization – New Vision, DHS/DOD Software Assurance Forum, March 8, 2007.
33. National Defense Industrial Association System Assurance Guidebook, Version 0.89. National Defense Industrial Association, System Assurance Committee, Arlington, Virginia February 2008.
34. IEEE Std 1471-2000 has been superseded by ISO/IEC/IEEE 42010-2011 - Systems and software engineering – Architecture description

The Whole Is More Than the Sum of Its Parts:

Understanding and Managing Emergent Behavior in Complex Systems

Dean M. Morris, Software Engineering Consultant
Kevin MacG. Adams, Ph.D., NCSOSE

Abstract. How does the project or maintenance manager control the unknown? The unknown in this case is the negative or positive behaviors or properties that emerge from a complex software system. The application of systems theory to software is becoming increasingly important as systems become more complex. Looking at a complex software system through the lens of systems science can give the manager the insight needed to understand and control negative or enhance positive emergent behaviors. This article provides an overview of some of the key terms and concepts of systems theory, complexity, and emergence. Both the positive and negative effects of emergent behavior on software systems are considered. Additionally, some speculative and new methodologies for managing undesirable emergent behavior are explored.

Introduction

Imagine a hypothetical scenario where you were contracted to manage the development and maintenance of a new multi-server, web-based software system for the Defense Finance and Accounting Service. The system was tested vigorously and passed all acceptance tests. After several months of operation, the system users noticed increasing lag in the system until it finally locked up. Tests of the individual components of the system indicated there were no problems. It only displayed the lock-up problem when the entire system was online and operating. So, what happened? Why did the system exhibit this negative behavior only after being in operation for some time?

A real-world variation of this scenario at another organization was presented by Mogul [1]. After much troubleshooting by the maintenance team, it was diagnosed that the database server load balancer was set incorrectly. As data was being received, routed, and stored in the databases, the databases' response time increased. The unexpected effect was the system load balancer interpreted the increased database delays as a failure. After the timing expectations of the load balancer were lowered (i.e. the expected response time from the servers was increased), the system functioned well.

In hindsight, it was concluded that the behavior of the load balancer was totally unexpected as it only manifested itself when the entire system was operating. This type of unexpected or emergent behavior in a complex system is one of the key concepts of systems theory [2]. As software systems have become much more prevalent in government, commercial, and peoples' daily lives, so too has the complexity of the systems that support them. The knowledge and application of systems theory and methodologies is becoming increasingly important for the management and maintenance of today's complex software systems.

A Brief Primer of Systems Theory and Emergence

Systems theory [3] provides the underlying theoretical foundation for understanding systems, and as such, serves as the foundation for the purposeful engineering for all complex systems. Knowledge of the systems theory axioms is essential for the modern software project or maintenance manager. While understanding all the basic concepts of systems theory is important in software systems, the concept of emergence or emergent behavior (both positive and negative) is paramount. By its nature, a software system usually only exhibits emergent behavior(s) after the system has been accepted and transitioned to the software maintenance phase. Thus, the thrust of this article is toward managing the maintenance of complex software systems with the potential to exhibit emergent behaviors.

Systems Theory

Systems theory is a unified system of propositions, linked with the aim of achieving an understanding of systems, while invoking improved explanatory power and predictive ability. It is precisely this group of propositions that enables thinking and action with respect to systems [3]. A theory does not have a single proposition that defines it, but is a population of propositions (a model) that provides a skeletal structure for the explanation of real-world phenomena. The relationship between theory and its propositions is not a direct relationship. It is indirect, through the intermediary of the axioms, where the links in the theory represent the correspondence through similarity to the empirical, real-world system. Figure 1 depicts these relationships.

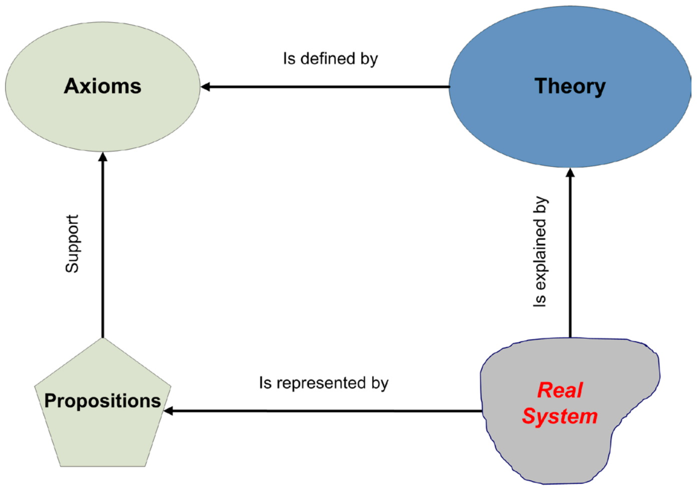


Figure 1: Propositions, Axioms, Theory and the Real World System

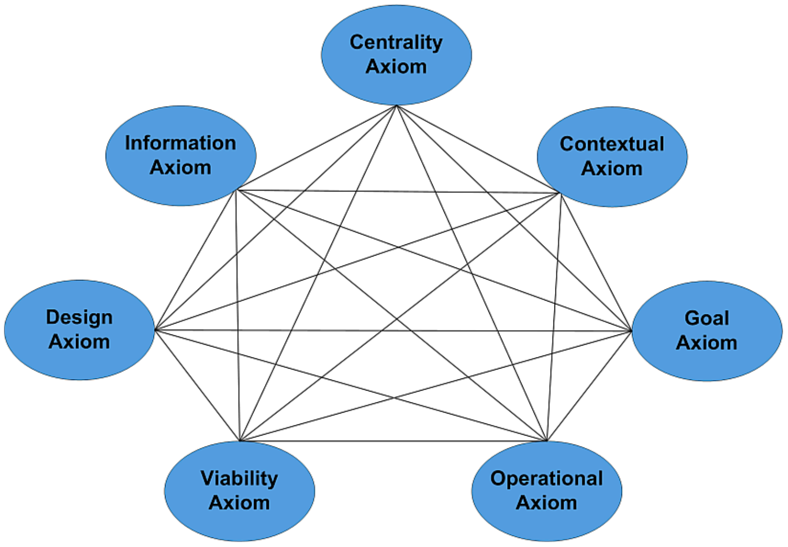


Figure 2: Axioms of Systems Theory

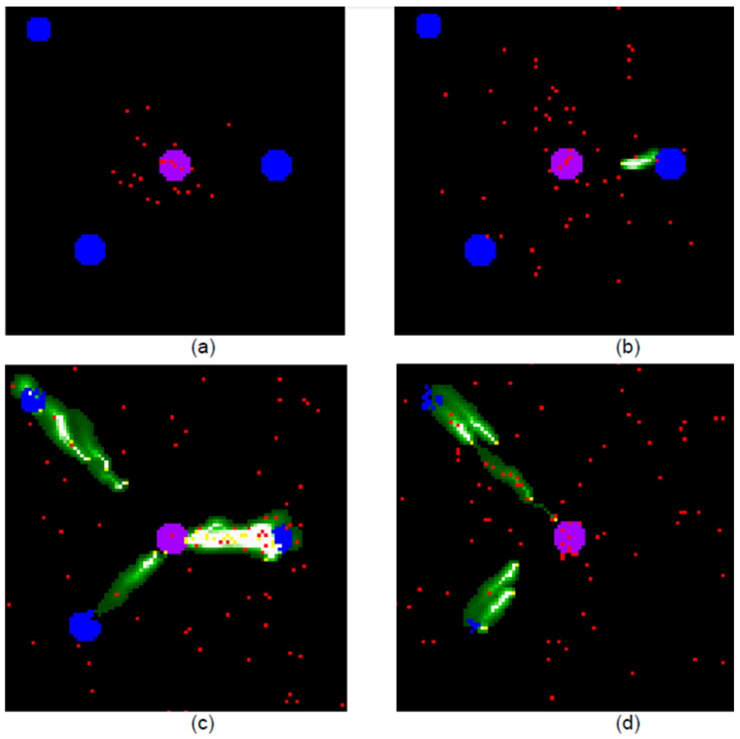


Figure 3: Snapshots from an Ant Colony Simulation [7]

Systems theory provides explanations for real world systems. The explanations increase our understanding and provide improved levels of explanatory power and interpretation for the real world systems we encounter. Our view of systems theory is a model of linked axioms that are represented through similarity to the real system [4]. Figure 2 is a model of the axioms of systems theory. The axioms presented in Figure 2 are called the theorems of the system or theory [5] and are the select set of propositions, presumed true by systems theory, from which all other propositions in systems theory are deducible.

The axioms of systems theory [6] are as follows:

• The Centrality Axiom states that central to all systems are two pairs of propositions; emergence and hierarchy, and communication and control.

• The Contextual Axiom states that system meaning is informed by the circumstances and factors that surround the system. The contextual axiom's propositions are those which give meaning to the system by providing guidance that enable an investigator to understand the set of external circumstances or factors that enable or constrain a particular system.

• The Goal Axiom states that systems achieve specific goals through purposeful behavior using pathways and means. The goal axiom's propositions address the pathways and means for implementing systems that are capable of achieving a specific purpose.

• The Operational Axiom states that systems must be addressed in situ, where the system is exhibiting purposeful behavior. The operational axiom's propositions provide guidance to those that must address the system in situ, where the system is functioning to produce behavior and performance.

• The Viability Axiom states that key parameters in a system must be controlled to ensure continued existence. The viability axiom addresses how to design a system so that changes in the operational environment may be detected and affected to ensure continued existence.

• The Design Axiom states that system design is a purposeful imbalance of resources and relationships. Resources and relationships are never in balance because there are never sufficient resources to satisfy all of the relationships in a systems design. The design axiom provides guidance on how a system is planned, instantiated, and evolved in a purposive manner.

• The Information Axiom states that systems create, possess, transfer, and modify information. The information axiom provides understanding of how information affects systems.

Emergence

Central to the discussion of system theory is the centrality axiom and the principles of emergence and hierarchy. Hierarchy and emergence contribute to complexity because new and interesting properties that cannot be found in the parts emerge and add a whole new dimension to understanding [2]. The father of modern systems theory, Ludwig von Bertalanffy explains that the meaning of the somewhat mystical expression, "The whole is more than the sum of its parts," is simply that constitutive characteristics are not explainable from the characteristics of isolated parts. The characteristics of the complex, therefore, compared to those of the elements, appear as new or emergent [7].

However, Odell [8] noted that complex systems do not have to be complicated to display emergent behavior. In fact, the agents (i.e., the elements) of the system can all be homogenous, follow a simple set of rules, and still exhibit emergent properties. To illustrate the point, Odell [8] described an ant colony computer simulation where each ant agent behaved by the following rules:

1. Wander randomly.
2. If food is found, take a piece back to the colony and leave a trail of pheromones that evaporate over time; then go back to rule 1.
3. If a pheromone trail is found, follow it to the food and then go to rule 2.

Figure 3 shows the simulation display at four stages. The anthill is represented by the purple circle in the center and the three blue dots are the food piles. The ants are the small red dots and their pheromone trails are the white and green areas.

Notice that in Figure 3.a, the ants have individually begun moving away from the colony in a random way. Later in the sequence, in Figure 3.b, some ants have located the food on the right and have started marking their path back to the anthill with pheromones as they carry bits of food. By the time of Figure 3.d, the entire right food pile has been moved to the colony and they are rapidly consuming the other two piles of food.

Relating back to systems theory, while the ants individually behave by a simple rule set, collectively as a colony system, they act in a complex way. They communicate with each other via the pheromone trails, while the pheromones also serve to exert overall system control. That is, when an ant encounters a pheromone trail, it is obligated to follow it to the food, get some food, and return to the anthill marking the path with more pheromones. The resulting emergent property for the ant colony system is a full food storage area. This emergent property cannot be discerned by observing the behavior (via the simple rule set) of individual ants. Only once the ants begin interacting in an environment where there is food and a storage area (i.e. the colony) does the emergent property of food storage become evident.

Effects of Complexity and Emergence on Software Systems

This section moves systems theory into the realm of software systems. Before getting into some positive and negative implications of complexity and emergence within software systems, a more abstract view of the software and its stakeholders will be discussed in a systems theory context.

Evolution of Software

Rajlich and Bennett [9] developed a versioned, staged model for software maintenance because they believed that the more traditional models of maintenance did not accurately capture the evolutionary nature of the software lifecycle. Rajlich and Bennett's [9] maintenance model consists of the following five stages:

1. Initial development: Not maintenance yet.
2. Evolution: Significant changes may be made to a given version to meet changing user needs. The experience of the development team and an adaptable architecture are being leveraged to accommodate the major changes. This stage is iterative for the given version.
3. Servicing: As team experience and knowledge for the version is lost and the code starts to decay; only minor updates are made to the system. This stage is also iterative.
4. Phaseout: No more updates are performed as the system continues to operate.
5. Closedown: The system is retired from service.

In this model, after the initial development, the first version enters the evolution stage where significant updates are made in an iterative fashion. Even as the current version is being supported through the various stages, the development team is evolving the system to the next major version that will enter its own set of stages upon release. For an example, Rajlich and Bennett [8] pointed out that the Microsoft Corporation uses this model for the production, evolution, and support of its operating systems (e.g. Windows XP, Vista, 7, etc.).

Examining this model from a systems perspective, a few observations can be made. Independent of whether or not the actual software exhibits emergent behavior in its operation, the fact that the system is being evolved both within each version and to the next version indicates that there is a complex system involved. The system where the evolution is occurring is at least one hierarchal level up from the software system and includes human agents (i.e. stakeholders such as developers, maintainers, users, etc.) interacting with each other. Also, the environment outside of the open system may be changing (e.g. competition with rival organizations, advancements in technology, etc.), thus causing the system to adapt and evolve.

Positive Emergent Behavior

Moving back down to the software system level, positive emergent behavior represents great potential. Ideally, a developer can design a system so that desired properties emerge while undesired behaviors can be suppressed. This is a difficult task, as emergent behavior is unpredictable by nature. In one research paper, Maciaszek [10] prescribed a meta-architecture for complex software systems that was known to produce the desirable emergent property of adaptability while preventing other properties from emerging. This strategy can be applied to using design patterns to repeat positive results from previous proven systems.

In other research, Olaru, Gratie, and Florea [11] developed a data distribution scheme using a cognitive Multi-agent System (MAS). The overall concept is that simple cognitive agents that have basic goals and behaviors are connected in a network or matrix configuration. Data can be introduced into the system through any of the agents. After the data is introduced into the MAS, it is propagated throughout the system so that it is available to be read from any agent in the system. Thus, individual cognitive agents interacting on the local level produce the emergent property of distributing the data throughout the system without any central control.

Negative Emergent Behavior

Even though positive emergent behavior in software systems holds great promise for the future, the maintenance manager or developer of today will more than likely have to deal with mitigating the undesirable or negative emergent behaviors in complex software systems. How does the maintainer troubleshoot and repair a problem that does not originate in the code, but instead originates in the interactions between agents in the system? Mogul [1] proposed a research agenda to come to grips with the negative emergent property (i.e. misbehavior) problem in the software industry. Agenda items include:

1. Creating a taxonomy of emergent misbehavior.
2. Creating a taxonomy of typical causes.
3. Developing detection and diagnosis techniques.
4. Developing prediction techniques.
5. Developing amelioration techniques.
6. Developing testing techniques.

Mogul [1] provided preliminary taxonomies of emergent misbehavior and typical causes in the paper, but indicated that the other four technique categories would be much more challenging to develop and implement. Work in these categories is ongoing as is evident with the following.

Managing Negative Emergent Behavior in Software

Software maintenance is already difficult enough in regular systems, let alone in complex software systems with emergent behavior. Software project managers can benefit by keeping up with the software industry literature to gain insight into potential methods for mitigating negative emergent behavior.

Toward Self-maintaining Systems

Pertaining to the software systems of the near future, Gabriel and Goldman [12] wrote:

"Future innovations in software will need to produce systems that actively monitor their own activity and their environment, that continually perform self-testing, that catch errors and automatically recover from them, that automatically configure themselves during installation, that participate in their own development and customization, and that protect themselves from damage when patches and updates are installed. Such systems will be self-contained, including within themselves their entire source code, code for testing, and anything else needed for their evolution."

To meet these goals, Gabriel and Goldman [12] proposed a hypothetical hybrid autopoietic and allopoietic system. According to Gabriel and Goldman [12], an autopoietic system is one that is continually re-creating itself and allopoesis is the process whereby a system produces something other than the system itself. In essence, the autopoietic part of the system would concentrate on keeping the system viable via monitoring the system health and taking corrective action if a system-threatening problem developed (e.g. a negative emergent behavior). The allopoietic part of the system would operate as programs do today (i.e., perform the functions of the system).

Verifying Complex Systems Through Formal Methods

NASA's answer for dealing with undesirable emergent behavior in a complex system may lie with verification through a formal methods cocktail. Rouff, Hinchey, Truszkowski, and Rash [13] reported on research into the viability of utilizing formal methods to verify the emergent behavior of the Autonomous Nano-Technology Swarm (ANTS) mission that may be used to explore the asteroid belt.

Basically, the mission entails 1,000 two-pound autonomous space vehicles that will be transported to the edge of the asteroid belt. From there the ANTS will self-organize into exploration teams with leaders. Various instruments will be used to collect data from asteroids that will be periodically transmitted back to earth. For autonomous operation, the ANTS will need to exhibit the properties of self-configuration, self-optimization, self-healing, and self-protection. Because the ANTS mission will potentially depend on certain positive emergent behaviors to operate while not developing any negative attributes, many formal methods and techniques were considered for the verification of this intelligent swarm. After the evaluation, the research team settled on a combination of four current formal methods that they plan to integrate into one method that is best suited to verifying the behavior of intelligent swarms. It is conceivable that a similar combination of formal methods could be used to verify other complex software systems to prevent negative properties from emerging while grooming desired emergent behaviors.

Repairing Emergent Behaviors Through Runtime Feedback

Lewis and Whitehead [14] have conceded that many emergent behaviors simply cannot be detected using testing or other verification techniques. To combat this problem, they developed a system for detecting and repairing undesirable emergent behavior at runtime. The main component of the system is a runtime monitor named Mayet. The program they experimented with was a variation of the game Super Mario Brothers.

For the system to work, rules were input into Mayet so it would know what undesirable behaviors to look for (e.g. the character gets stuck in an on-screen object, jumps too high, etc.). Upon detecting an error, Mayet sends a message to the game. After the game receives the message, the game repairs the problem almost instantaneously. A major catch is that the repair routines have to be built into the game. This seems problematic because the developer has to anticipate the possible repairs that may be required while the types of behavior that are supposed to be fixed are emergent, thus difficult to predict. Regardless, the concept of repairing an emergent problem in a software system as it is operating is a step in the right direction.

Conclusion

The knowledge and application of systems theory and methodologies has become increasingly important for the management and maintenance of today's increasingly complex software systems. The promise and the problem of emergent behavior in complex software systems is a double-edged sword. Those that chose to ignore the implications of systems science and emergent behavior will be relegated to a reactionary role. Project and maintenance managers who embrace the systems science viewpoint will be much better prepared to be proactive in controlling their software systems. The government and civilian software engineering communities will need to gain a deeper understanding of how to capitalize on the synergies that positive emergent properties can provide while reliably excluding negative emergent behaviors from software systems.

ABOUT THE AUTHORS

Dean M. Morris is a software engineering consultant who recently developed software supporting research at the National Centers for System of Systems Engineering (NCSOSE). He retired from the Air Force after serving in the communications-electronics maintenance field for 21 years. Morris holds a B.S. in Computer and Information Science (with a minor in Finance) and an M.S. in Software Engineering from the University of Maryland University College.

2765 Hamilton Road
Waldorf, MD 20601
E-mail: deanmorris@ieee.org

Dr. Kevin MacG. Adams is a Principal Research Scientist at the National Centers for System of Systems Engineering (NCSOSE). Dr. Adams is a retired Navy submarine officer and information systems consultant. He was on the faculty at Old Dominion University from July 2007 until coming to NCSOSE in January 2009. Dr. Adams holds a B.S. in Ceramic Engineering from Rutgers University, an M.S. in Naval Architecture and Marine Engineering and an M.S. in Materials Engineering both from MIT, and a Ph.D. in Systems Engineering from Old Dominion University.

4111 Monarch Way, Suite 406
Norfolk, VA 23508-2563
E-mail: kmadams@odu.edu

REFERENCES

1. Mogul, Jeffrey C. "Emergent (Mis)Behavior vs. Complex Software Systems." ACM SIGOPS Operating Systems Review 40.4 (2006): 293-304.
2. Flood, Robert, and E. Carson. Dealing with Complexity: An Introduction to the Theory and Application of Systems Science (2nd Ed.). New York: Plenum Press, 1993.
3. Adams, K. MacG. "Systems Principles: Foundation for the SoSE Methodology." International Journal for System of Systems Engineering 2.2/3 (2011): 120-55.
4. Giere, Ronald N. Explaining Science: A Cognitive Approach. Chicago: University of Chicago Press, 1988.
5. Honderich, T., ed. The Oxford Companion to Philosophy (2nd Ed.). New York: Oxford University Press, 2005.
6. Adams, K. MacG., Hester, P. T., Meyers, T. J., Bradley, J. M. and Keating, C. B. Systems Theory as the Foundation for Understanding Systems (NCSOSE Position Paper 2012-001). Norfolk, VA: National Centers for System of Systems Engineering, 2012.
7. Bertalanffy, L. von. General System Theory: Foundations, Development, Applications (Rev. Ed.). New York: George Braziller, 1968.
8. Odell, James. "Agents and Complex Systems." Journal of Object Technology 1.2 (2002): 35-45.
9. Rajlich, V. T., and K. H. Bennett. "A Staged Model for the Software Life Cycle." Computer 33.7 (2000): 66-71.
10. Maciaszek, Leszek A. "Modeling and Engineering Adaptive Complex Systems." Tutorials, posters, panels and industrial contributions at the 26th international conference on Conceptual modeling - Volume 83. 1386961: Australian Computer Society, Inc., 2007.
11. Olaru, Andrei, Cristian Gratie, and Adina Magda Florea. "Emergent Properties for Data Distribution in a Cognitive Mas." Computer Science & Information Systems 7.3 (2010): 643-60.
12. Gabriel, Richard P., and Ron Goldman. "Conscientious Software." ACM SIGPLAN Notes 41.10 (2006): 433-50.
13. Rouff, Christopher, et al. "Experiences Applying Formal Approaches in the Development of Swarm-Based Space Exploration Systems." International Journal on Software Tools for Technology Transfer (STTT) 8.6 (2006): 587-603.
14. Lewis, C., and J. Whitehead. "Repairing Games at Runtime or, How We Learned to Stop Worrying and Love Emergence." Software, IEEE 28.5 (2011): 53-59.

Developing a Model for Simplified Higher Level Sensor Fusion

Mike Engle, The George Washington University
Shahram Sarkani, The George Washington University
Thomas Mazzuchi, The George Washington University

Abstract. Mulitsensor data fusion (MSDF) has been researched for decades yet programs relying on it to provide a situational, or threat, assessment continue to be less than successful. In order to alleviate the too-much-information, too-few-analysts issue, a better approach must be determined. A survey of recent and current data fusions programs was conducted along with a literature review on how different organizations handle a fusion-based assessment. Key points found in this study were used to develop an adaption of models that can be used to provide an improved assessment while simplifying the process needed to get there.

MSDF programs have long been a goal of the DoD and the warfighter. It promises to combine information from multiple sensors in order to determine what traditionally could not be determined by one sensor alone either because of technological limitations or geographic restrictions. These multisensor systems can be used to increase geolocation accuracy, reduce uncertainty, automatically extract man made features, and quickly identify potential targets. When expanded to include higher-level fusion capabilities, MSDF tools can help anticipate future actions of these potential targets or provide recommendations for anticipated decision points. It is no longer enough to simply provide image registration or to combine sensor level information when higher-level fusion based software promises improved situational awareness and autonomous decision-making aids.

The demand for MSDF systems has only increased in the era of near ubiquitous sensors. With more sensors, especially with the move into persistence, come more data and the need for more analysts to review the data. The problem has long since arrived that there is too much data for too few analysts. The goal is not to replace the analyst but to better enable them to use the information that is already available. How often has the world been surprised by a significant incident only later to find that there were indicators available to prevent it? Events like the 2009 Christmas Day Bombing or the Ft Hood shooting were preceded by sufficient indicators; all that was needed was someone to piece together the parts in a timely manner.

It should already be clear why data fusion has been researched for decades. Still today, there are dozens of contractors and universities dealing with multiple agencies who continue searching for solutions [1, 2]. The National Geospatial-Intelligence Agency (NGA) outreach lists multi-source and multi-INT fusion as priority for research and has asked for help in tackling what they consider a hard problem [3]. In fact, the NGA has increased research into different fusion technologies to such an extent that other agencies have reduced their funding [4].

Unfortunately, many of these fusion programs have been less than successful and the golden age of sensor fusion has not yet arrived [3, 4]. Several factors can be attributed to this issue. At the sensor level, these systems must combine data with varying temporal, spatial, spectral and radiometric characteristics. They, "may be heterogeneous, possibly asynchronous, and not identically georeferenced due to motion, limited fields of view, or constraints on power and/or the GPS signal [4]." At the program level, problems have arrived from too grand a goal to start with, the requirements of a wide range of disciplines not traditionally used in systems or software engineering, and the use of what are traditionally very stove-piped, isolated tradecraft.

Sensor Fusion Defined

There may be as many interpretations about what defines data fusion as there are people who are trying to solve it. The sensor fusion domain not only includes combining the outputs of single-modal, single-phenomenology sensors but also the predictive assessments provided by systems relying on multi-platform (different unmanned aerial vehicles for example), mult-INT (combining multiple intelligence types such as imagery intelligence and electronic signals derived data. An instructive way to define DF while conveying its wide scope is to use a process model. The most referenced model within the DoD appears to be the Joint Director of Labs (JDL) data fusion model shown in Figure 1 [5].

The JDL, is an organization which no longer exists but in the 1980s they were tasked to develop a model for data fusion. This JDL model, revised in 1999, was created to show a general process of data fusion with wide applicability for both government and academia. It standardizes communications between engineers but does not dictate the actual steps of performing fusion nor which levels must be used. The model shows multiple potential data sources on the left that can be directed to any of a number of processes within the fusion domain then the resulting output provided on the right. Table 1 provides a description of the most common fusion levels [6].

As an example, detecting a manmade object at a specific location, classifying it as a tank and even identifying it specifically as a T-72 tank is all covered under Object Assessment (level-1). Situation assessment (level-2) can use priory information to indicate that this Soviet-designed main battle tank is possibly a friendly unit of the Iraqi Army. The number found and location would further indicate unit size, if not the exact unit, and possibly the unit's disposition such as movement to contact. The impact assessment (level-3) could use this information then indicate that the explosions detected by acoustic sensors may not have been an attack directly on coalition forces; however units should be moved to support the Iraqi Army.

Though this model does not indicate a process where one level must be met before the next, programs traditionally start at level-1 then determine what must be accomplished in order to reach the next level on up. This has led to a large number of programs which have worked through level-1 processing while not too many have successfully developed level-3 [2]. When working through each level in this manner level-3 becomes an increasingly complex goal. This complexity is further increased when the need for increased situational awareness necessitates moving from the fusion of different single-INT sensors to the fusion of different multi-INT sensors.

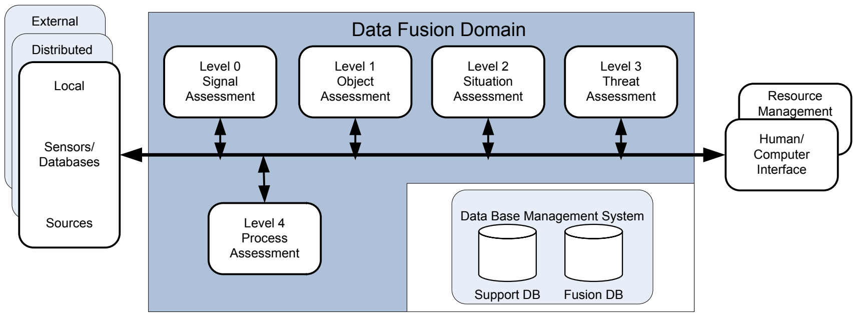

External
Distributed
Local
Sensors/ Databases
Sources
Data Fusion Domain
Level 0 Signal Assessment
Level 1 Object Assessment
Level 2 Situation Assessment
Level 3 Threat Assessment
Level 4 Process Assessment
Data Base Management System
Support DB
Fusion DB
Resource Management
Human/ Computer Interface

Figure 1 The revised JDL Data Fusion Model (Hall, Liggins, & Llinas, 2009)

Level	Name	Description
0	Subobject/ Signal Assessment	Preconditioning Data to correct biases, perform spatial and temporal alignment. Also can include feature extraction or signal detection.
1	Object Assessment	Association of data to estimate an objects or entity's position, kinematics, or attributes (including identity)
2	Situation Assessment	Aggregation of objects/events to perform relational analysis and estimation of their relationships in the context of the operational environment.
3	Impact /Threat Assessment	Projection of the current situation to perform event prediction, threat intent estimation, own force vulnerability, and consequence analysis.
4	Process Assessment	Evaluation of the ongoing fusion process to provide user advisories and adaptive fusion control or to request additional sensor/source data (resource management)

Table 1 JDL Fusion Levels 1-4 with descriptions

An Analysis of Recent Projects

An existing initiative already offers a concise summery of current technology-based programs, the National Technology Alliance (NTA). One of the benefits provided by the NTA is simplifying USG access to commercial technology; specifically dual-use technology where cost-sharing can be attained. It also provides an independent assessment and evaluation of government users' needs and identifies optimum technology solutions to technical challenges [1]. Several of these analyses have covered data fusion research but in 2009 the multi-source and multi-INT Fusion Technology Survey and Analysis report conducted in conjunction with the Pennsylvania State University directly aligns with the type of work needed. Though this report covered several hundred government and COTS sensor fusion solutions, 24 separate projects ranging from basic research to tool development were picked for additional study. These projects represent the work funded by a single R&D office whose goal was the advancement of available sensor fusion based tools.

First, information was collected from their project summaries as a starting point to show the breakdown of what was included in this sample space. Then a more in-depth analysis into each project was made in order to provide an independent look while ensuring each was evaluated by a single person. This was done to remove any bias or at least provide a consistent bias across all 24 projects. Finally, a third look was attempted after approximately one year in order to determine a status update.

The assumption was that the majority of the programs would concentrate on both a single intelligence gathering discipline (INT) and lower-level sensor fusion techniques. Once the information was collected, 11 of these selected projects concentrated on a single INT although most did span across multiple phenomenologies. Seven programs were described as multi-INT, while most of these simply provided a common geospatial reference to a specific type of non-geospatial intelligence data. The remaining six included support items such as database development and were determined to not be directly applicable to this breakdown. Figure 2 shows that of the 18 represented projects, nine were considered level-1 fusion, seven were considered level-2 and the final two were considered level-3 fusion.

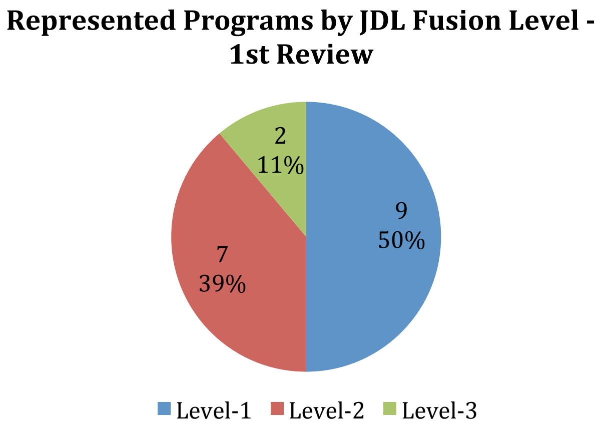


Figure 2 Represented programs broken down by JDL fusion level IAW project description

The independent audit of the 18 represented projects showed that a total of 15 were likely level-1 fusion technologies. This left only one of the original seven level-2 projects in place to support situational assessment. The two projects originally indicated as level-3 fusion remained level-3 (Figure 3). Of these final two, one turned out to be a study. This study was not rejected as a level-3 project because it potentially laid out important groundwork for follow-on multi-INT work. However, it did not provide for any actual data fusion in itself. This left a single project out of a total of 24 to possibly become a higher-level data fusion based software tool.

During the review several issues were noted. It was found that that a large percentage of the level-1 fusion projects required multiple separate hard problems to be answered in order to be successful. Some of these problems were the same but approached separately between the separate projects and were therefore redundant efforts. In one instance a problem was worked though using a supporting technology that was known to be untested and at a very low technology readiness level. Though there was testing as part of the normal tool development process, none of it was meant to test performance of individual technologies before being integrated into the tool. This shows that projects were initiated without determining existing capability gaps and continued using high-risk methodologies

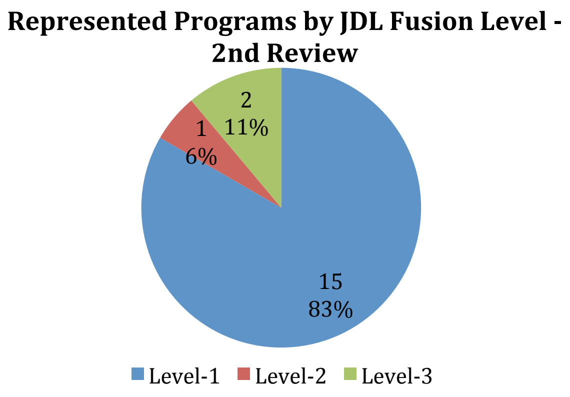


Figure 3 Represented programs broken down by JDL fusion levels determined by independent audit

After all work was initially planned to be complete a third round of review was undertaken. This review was less successful. It was not possible to find the exact status of any individual project the organization was working on. It was only possible to find artifacts of work leaving the organization. This included projects being sent out for independent testing or transitioning to a semi-operational status. From what was found, the initial 24 projects were roughly correlated to only three available MSDF tools. These conclusions also support previously cited literature stating that these types of programs tend to mostly be lower-level data fusion based with few successful higher-level programs.

Evaluating Alternative Approaches to Data Fusion

After reviewing the types of existing sensor fusion programs, the next step was to evaluate the process other organizations used to attain what could be interpreted as level-3 data fusion. Areas covered included legacy military, finance, and weather projects. This investigation converged on one manually intensive procedure that closely parallels the MSDF process discussed earlier. It is described in the Army's Field Manual on Intelligence, FM 2-0 [7].

The Army defines a procedure through the military decision making process (MDMP) to help identify the most important information to a commander. This is important because it is likely that there will always be too much information available and the commander does not need to track the status and update from each individual information source. FM 2-0 includes this process in the key intelligence task "conduct ISR" summarized in Figure 4. This is an exhaustive and iterative procedure that involves several key personnel with an in-depth understanding of the environment, unit capabilities, and what needs to happen to affect mission success.

This process starts with an understanding of the mission that needs accomplished. Then different courses of action are developed which are analyzed against the threat and environmental factors to produce a set of intelligence requirements and Priority Intelligence Requirements (PIR). Information deemed sufficiently important but not necessarily mission impacting are Intelligence Requirements. Information on hostile forces essential to support key decisions that must be made in order to accomplish a mission is classified as PIR. The process continues with an analysis of all available ISR (intelligence, surveillance and reconnaissance) assets and their capabilities. This, along with the initial MDMP, helps identify collectable indicators of threat intentions and objectives which can then be used to task subordinate units and ISR collection platforms.

Combining New Technology with Proven Process

A method to simplify building a set of fusion algorithms to take into account any number of sensory input and to try to think through possibly infinite scenarios is to start at the traditional end point (level-3 DF) to determine what actually needs to be assessed then move backwards by determining what must be obtained in order to get what is needed. In other words, if the traditional progression of data fusion is reversed and combined with the Army's Intelligence Synchronization discussed in the previous section, then a skeleton process of simplified multisensor data fusion starts to take shape (Figure 5).

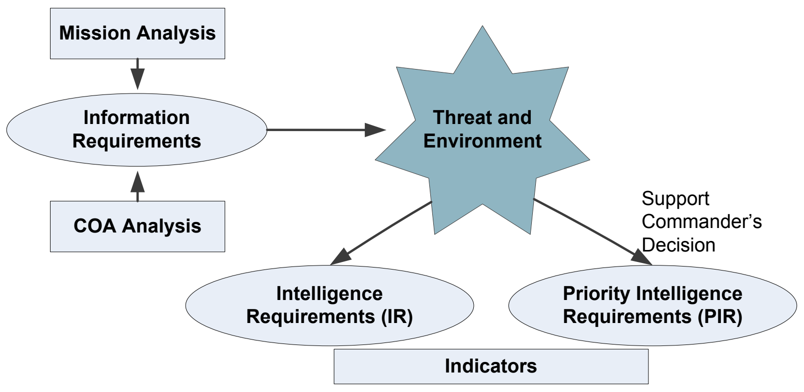

Mission Analysis
Information Requirements
COA Analysis
Threat and Environment
Support Commander's Decision
Intelligence Requirements (IR)
Priority Intelligence Requirements (PIR)
Indicators

Figure 4 Army ISR Task Development Process from FM 2.0 which takes the mission, threat and environment into account to determine the most significant intelligence requirements

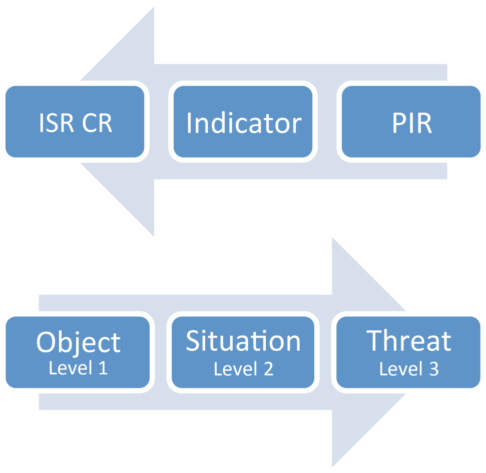


Figure 5 The Reverse Data Fusion Model shown over the initial JDL fusion model with traditional workflow indicated

Taken a step further, IRs can be analyzed using knowledge of the organization's existing intelligence capabilities to determine which could be met using an automated fusion process. These would be labeled as Fusion Information Requirements (FIRs). FIRs are the intelligence requirements that can be autonomously processed by current and potential sensor and used in fusion processing. These FIRs are broken down into indicators that support the FIRs and can be labeled as facts. These facts are the actual observations that can be detected by any of the available intelligence sensors and matched against priory information. In other words, these indicators are used to support any one of a number of situation assessments that have been predetermined as necessary in order to match a threat assessment or answer a PIR.

Next, the most likely methods to observe these indicators are thought through. Each may have multiple methods of detection. Depending on timeliness requirements, available sensors and the environment, each reasonable detection is used to create a Collection Requirement (CR). CRs are the tasking to the specific intelligence collector such as aircraft, soldiers or ground sensors that are most likely to observe what is needed in the time frame that it is needed. Each CR is added to the existing requirements management process. An example is shown in Figure 6 where three separate FIRs are broken down into their applicable facts. FIR-1 needs three Facts meet in order to be satisfied. Each fact can be met through a set of detections using Boolean logic.

Figure 7 shows how the FIRs (previously PIRs), Facts (indicators), and CRs loosely align with but move opposite of the more traditional object, situation, and threat assessment functions of the JDL MSDF model. This process continues in cycles as the threat evolves, new PIRs are determined, or the availability of different ISR platforms change. This creates both a synchronized collection effort and a modular approach to MSDF. It also provides a basis for real-time information collection and processing without creating any redundant processes to an organization. Even if the result is only an alert in an operations center or an email sent to the responsible analyst, pertinent and timely information is sent to the specific person in need, in near real-time, without having to monitor countless hours of data feeds.

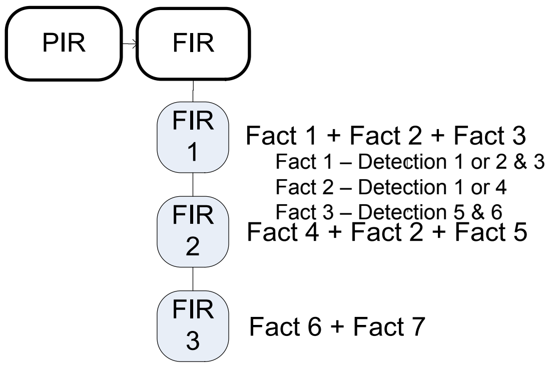


Figure 6 PIRs that can be used as fusion information requirements are further broken down into Facts and Detections

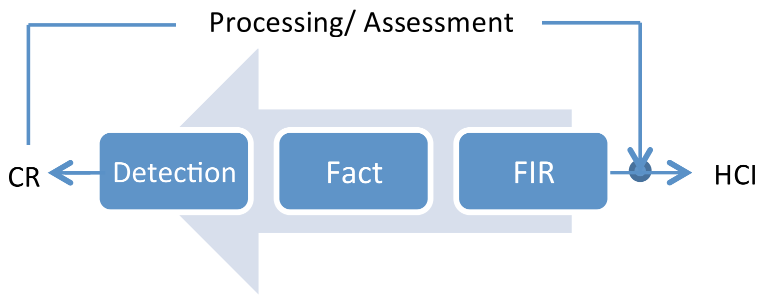


Figure 7 Reverse higher level data fusion model

Conclusion

Higher level multisensor data fusion programs allow for a solution that is more significant than the sum of the data supplied to them. In this case, they take new and known information and provide a level of data abstraction in order to help understand what is going on and to do this quicker then what would normally be possible. This allows for timely decisions to be made as events occur or statuses change, instead of after analysts have had time to analyze each situation manually.

Care should be taken to limit work that is too similar to work already funded or completed. This includes anything from basic R&D initiatives to acquisition programs placing major end items into combat. Care should also be taken to limit the overall scope of what a MSDF program may cover. If the intent is to develop a new MSDF system for a specific purpose then do not add new and unrelated capabilities. Many very capable systems already exist but varying missions and the effects of rapid fielding initiatives have limited their capabilities and interconnections into other systems. Using principles of modular systems engineering and borrowing from aspects of different levels of sensor fusion (fusion at the sensor, object or decision level) and a simplified method of improving the common operation picture may be possible while leveraging on existing capabilities.

Though many MSDF programs have met with limited success it seems entirely possible that simply reversing the order in which most programs run may affect positive outcomes. By taking what absolutely must be known (facts), finding ways to first characterize then indicate these facts through available sensor detections, then to provide an output largely based on relatively simple Boolean math and a whole new model for future programs is created. When taken in the traditional lower to higher level DF order, advanced processing must be developed in order to account for countless possible combinations of unknown future indications. The reverse model alleviates the need of this advanced methodology, such of cognitive engineering and neural networks, and simply waits for detections that can answer the commander's priority information requirements.

ABOUT THE AUTHORS

Mike Engle has a Bachelor of Science in Mechanical Engineering from the Pennsylvania State University and a Master's of Science in Systems Engineering from the George Washington University. He has provided systems engineering support to a variety of R&D and software engineering organization over the past 10 years. Prior to graduate school, Mike Engle was a US Army aviation officer.

The George Washington University
1776 G Street, NW Suite 101
Washington, DC 20052
Phone: 703-599-0624
E-mail: tme110@gwu.edu

Dr. Shahram Sarkani joined the faculty of the School of Engineering and Applied Science (SEAS) at The George Washington University in 1986. He currently serves as the faculty advisor for Off-Campus Programs in the Department of Engineering Management and Systems Engineering. From 1994 to 1997, he served as chair of the Civil, Mechanical, and Environmental Engineering Department. From 1997 to 2001, he was SEAS interim associate dean for Research and Development. Dr. Sarkani holds a BS and MS in Civil Engineering from Louisiana State University and a PhD in Civil Engineering from Rice University.

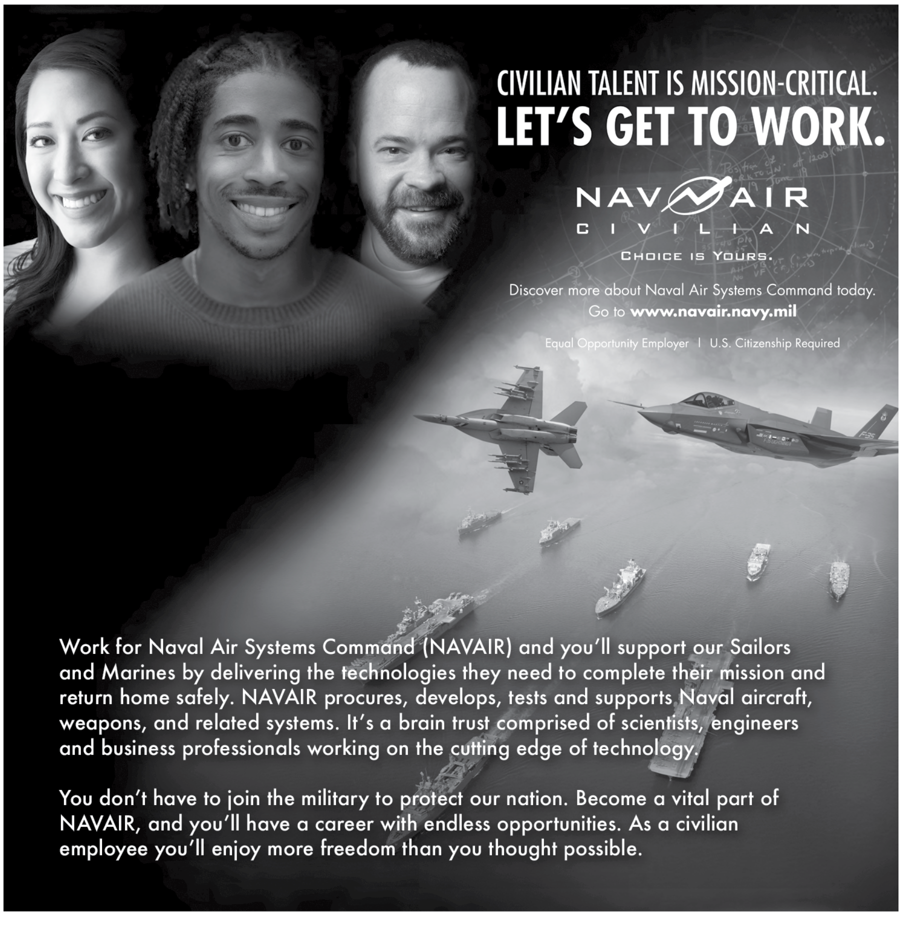
The George Washington University
1776 G Street, NW Suite 101
Washington, DC 20052
E-mail: sarkani@gwu.edu

Dr. Thomas Mazzuchi is a professor of Operations Research and Engineering Management at The George Washington University. His current research interests include reliability and risk analysis, Bayesian inference, quality control, stochastic models of operations research, and time series analysis. Dr. Mazzuchi earned a BA in Mathematics from Gettysburg College, and an MS and DSC in Operations Research

The George Washington University
1776 G Street, NW Suite 101
Washington, DC 20052
E-mail: mazzu@gwu.edu

REFERENCES

1. National Technology Alliance. (2009). Rosettex NTA project portfolio. No. TR-001-072709-554). Retrieved 13 March 2011 from <http://www.rosettex.com/nta/Rosettex%20Project%20Portfolio%20Sept%202009%20v1-1a.pdf>.
2. National Technology Alliance. (2009). Multi-source and multi-INT fusion technology survey and analysis, version 3. No. FR-001-085-052609-538PR). Retrieved 13 March 2011 <http://portal.opengeospatial.org/files/?artifact_id=38981>
3. NGA. (2009). Cooperative research and development agreement (CRADA) handbook. No. 10-008). Va: National Geospatial Intelligence Agency.
4. National Research Council. (2006). Priorities for GEOINT research at the national geospatial-intelligence agency. Washington, DC: The National Academies Press.
5. Hall, D. L., Liggins, M. E., & Llinas, J. (2009). Handbook of multisensor data fusion : Theory and practice (2nd ed.). Boca Raton, FL: CRC Press.
6. Hall, D. L., & Llinas, J. (2002). An introduction to multisensor data fusion. Proceedings of the IEEE, 85, 6-23.
7. Department of the Army. (2008). FM 2-0 intelligence (C1 ed.). Washington, DC: Department of the Army.


CIVILIAN TALENT IS MISSION-CRITICAL.
LET'S GET TO WORK.
NAVAIR
CIVILIAN
CHOICE IS YOURS.
Discover more about Naval Air Systems Command today.
Go to www.navair.navy.mil
Equal Opportunity Employer | U.S. Citizenship Required

Work for Naval Air Systems Command (NAVAIR) and you'll support our Sailors and Marines by delivering the technologies they need to complete their mission and return home safely. NAVAIR procures, develops, tests and supports Naval aircraft, weapons, and related systems. It's a brain trust comprised of scientists, engineers and business professionals working on the cutting edge of technology.

You don't have to join the military to protect our nation. Become a vital part of NAVAIR, and you'll have a career with endless opportunities. As a civilian employee you'll enjoy more freedom than you thought possible.

Basing Earned Value on Technical Performance

Paul Solomon, PMP

Abstract. Previous articles in CrossTalk and the Journal of Software Technology provided practical guidance to improve the quality of Earned Value Management (EVM) information [1, 2, 3, 4]. This update recommends contract language and project monitoring techniques to ensure that contractors integrate technical performance, including software functionality, with EVM. The key enablers are the Integrated Master Plan (IMP) and linkage to Systems Engineering (SE) work products and best practices.

EVM can become an effective program management tool and deliver better value to the taxpayer and warfighter if contractors revised their processes and reports to integrate technical performance and quality with cost and schedule performance. However, there are no contractual requirements within the acquisition regulations or contract data requirements to require that contractors:

1. Tie the technical baseline to the EV Performance Measurement Baseline (PMB).
2. Tie technical progress to the Technical Performance Measures (TPM) of the program, including progress towards achieving planned functionality.

EVM Challenges

The guidance in this article meets EVM challenges that were addressed in the DoD report to Congress; DoD EVM: Performance, Oversight & Governance Report that was required by the "Weapon Systems Acquisition Reform Act of 2009." The challenges concern technical performance and SE, as follows.

Technical Performance

- EVM can be an effective program management tool only if it is integrated with technical performance
- The engineering community should establish TPMs that enable objective confirmation that tasks are complete;
- If good TPMs are not used, programs could report 100% of earned value (or credit for work performed), even though they are behind schedule in terms of validating require ments, completing the preliminary design, meeting weight targets, or delivering software releases that meet the requirements.
- The EV completion criteria must be based on technical performance, the quality of work must be verified, and criteria must be defined clearly and unambiguously.
- The PM should ensure that the EVM process measures the quality and technical maturity of technical work products instead of just the quantity of work performed.

SE Process and Products

EVM can be an effective program management tool only if the:

- EVM processes are augmented with a rigorous SE process
- SE products are costed and included in EVM tracking.

If the SE lifecycle management method is integrated with the planning of the PMB, then EVM will accurately measure technical performance and progress.

Contractual Impediments to Effective EVM

Neither the Defense Federal Acquisition Regulation Supplement (DFARS) nor the Data Item Descriptions (DID) require contractors to tie EV to technical performance. The DFARS Earned Value Management System (EVMS) clauses cite compliance with the ANSI-748 EVMS guidelines. However, the use of TPMs is optional per EVMS. Per the defense acquisition program support methodology, "EVMS has no provision to measure quality. Use TPMs to determine whether your percent completion metrics accurately reflect quantitative technical progress and quality toward meeting key performance parameters."

EVMS focuses on the work scope and is silent on product scope. It also states, "EV is a direct measurement of the quantity of work accomplished. The quality and technical content of work performed is controlled by other processes." These loopholes create a "quality gap." The quality gap enables contractors to submit misleading management information. EV and the cost performance may be overstated when it is based on the percentage of drawings or code completed without regard to the technical maturity of the evolving design. As a result, the estimate at completion may be understated.

Useful guidance to link EVM with TPMs, the technical baselines, IMP accomplishment criteria, and SE work products is found in many DoD guides, as summarized at <http://www.pb-ev.com/Pages/DoDGuidance.aspx>. However, acquisition managers are not able to implement this guidance if the contractors fail to provide needed information. Even the IMP is optional in DoD guidance and not contractually required in DFARS.

Better Buying Power: Suppler Incentives

The DoD is striving to deliver better value to the taxpayer and warfighter by improving the way the it does business via the Better Buying Power (BBP) initiatives. BBP 2.0 includes the initiative to, "Institute a superior supplier incentive program." To support that initiative, the Navy is currently developing a pilot program for DoD with the intent to recognize and reward contractors who demonstrate superior performance by focusing on cost, schedule, performance, quality, and responsiveness.

The following opportunities and solutions should be considered when developing BBP 2.0 supplier incentives.

Opportunities and Solutions

The following guidance seizes four opportunities that underlie the EVM challenges, as shown in Figure 1. Solutions to improve contractual requirements and acquisition management follow.

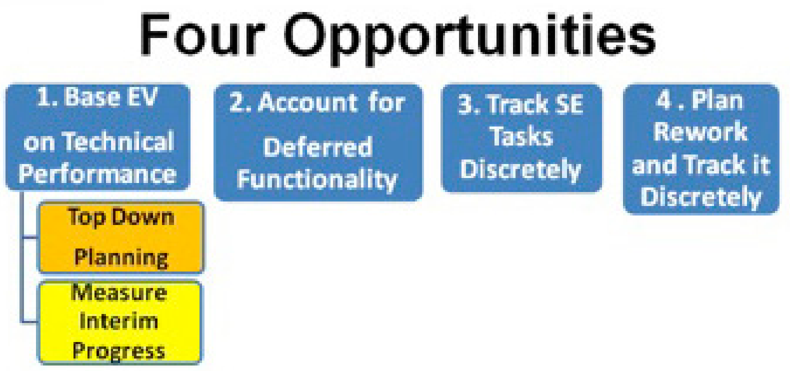

Four Opportunities
1. Base EV on Technical Performance
2. Account for Deferred Functionality
3. Track SE Tasks Discretely
4. Plan Rework and Track it Discretely
Top Down Planning
Measure Interim Progress

Figure 1

Base EV on Technical Performance

This opportunity has two components. First, do top down planning that includes defining milestones for achieving technical objectives. Then measure interim progress towards those meeting those objectives.

Top Down Planning

The solution for basing EV on technical performance has two components. First, develop integrated plans from the top down, starting with the technical baseline. Second, track progress towards meeting technical objectives.

The elements of effective, top down planning are:

1. Contractually-required IMP.
2. Use the Integrated Baseline Review (IBR) to reach agreement on IMP accomplishment criteria and to verify that contractor integrates technical performance and SE work products with the Integrated Master Schedule (IMS) and EVM.
3. Use major technical reviews and EVMS compliance reviews to verify that contractor maintains traceability from IMP to IMS to Control Account/Work Packages.

First, make the IMP a contractual requirement with requirements-based accomplishment criteria that are tied to the technical baseline. The criteria should include the completion of performance measures such as Measures of Effectiveness (MOE), Measures of Performance (MOP) and TPMs at key IMP events such as the System Functional Review (SFR), Preliminary Design Review (PDR), and Critical Design Review (CDR). Examples of accomplishment criteria are shown in Figure 2.

Second, use the IBR to forge agreements and to verify the degree of integrated program management. Verify implementation of the following during IBR:

- Requirements traceability from the requirements data base to the IMS and from the IMS to work package completion criteria.
- IMS includes interim and final milestones for development of SE work products with criteria that are consistent with the Contract Work Breakdown Structure (CWBS). The milestones include derived requirements, definition of required functionality and quality attributes, and verification methods and criteria.
- Milestones for establishing product metrics. MOEs and MOPs are defined at the SFR. TPMs are defined at the PDR.
- Milestones with technical maturity success criteria including TPM planned values, meeting requirements, and percent of designs complete.
- Define success criteria for event-driven technical reviews/IMP events.
- Revise/clarify criteria for CDR and subsequent events based on knowledge of revised and derived requirements to be met and TPM planned values.
- Flow down of SE milestones to work packages.

Measure Interim Performance

The solution to basing EV on interim, technical performance includes the following actions. First, establish objective linkage between TPM planned values and EVM. For physical objectives, use TPMs. For planned functionality, base EV on achieved functional requirements.

Next, compare the EV schedule variance (converted to duration) with the technical performance schedule variance. If the variances are inconsistent, perform a root cause analysis to determine reasons for the inconsistency. Then revise EV to be consistent with technical performance.

If technical performance is behind schedule, perform variance analysis and develop corrective actions. Then, revise the estimate to complete forward for work packages with corrective actions.

SFR	PDR	CDR
Functional Baseline	**Allocated Baseline**	**Product Baseline**
	Accomplishment Criteria	
1. Completed definition of the required system functionality • Functional and interface characteristics of overall system • Verification required to demonstrate their achievement includes • Detailed functional performance specification for the overall system • Tests necessary to verify and validate system performance. 2. Completed definition of MOEs and MOPs 3. All definitions above statused as complete in Requirements Data Base.	1. Completed definition of the configuration items (CI) making up a system • All functional and interface characteristics allocated from the top level system or higher-level CIs • Derived requirements • Performance of each lower level CI in the allocated baseline • Tests necessary to verify and validate CI performance. A technical performance baseline is in place down to the subsystem level, from which the system performance thresholds can be compared and tracked 2. All TPMs defined and allocated to Interface Control Documents and sub-systems). 3. All Key Performance Parameters (KPP), MOPs, and MOEs allocated to sub-systems 4. All definitions above statused as complete in Requirements Data Base	1. Completed definition of the required system functionality • Functional and interface characteristics of overall system • Verification required to demonstrate their achievement includes • Detailed functional performance specification for the overall system • Tests necessary to verify and validate system performance. 2. Completed definition of MOEs and MOPs 3. All definitions above statused as complete in Requirements Data Base.

Figure 2. Specified IMP Reviews, Baselines, Accomplishments/Criteria

Finally, correct EV to reflect the technical performance status. A backwards adjustment to EV is appropriate for work packages with corrective actions. This technique enables the use of EV to track corrective actions to resolution and closure.

Account for Deferred Functionality

In practice, contractors seldom account for deferred functionality when functional requirements are deferred from one build, release, or block to another. Normally, the numbered build and its respective work package are "closed" and 100% of the EV is taken, based on being finished with the build. When this happens, EV fails to disclose the true schedule variance. Also, cost performance is overstated.

The solution is to account for deferred functionality. If the build is released short of its planned functionality, the preferred technique is to take partial EV and close the work package. Then, transfer the deferred scope and Budgeted Cost of Work Remaining (BCWR) to the first month of the work package of the next increment. When this is done, EV mirrors technical performance and the schedule variance is retained.

Track SE Tasks Discretely

SE tasks are sometimes incorrectly planned as level of effort. Even when SE is discretely planned, EV is often based on interim milestones of progress towards completing a document such as a specification. These techniques fail to show objective progress towards completing requirements-based SE tasks such as requirements analysis and validation, definition of technical measures, or completion of trade studies. Getting behind schedule on these tasks is an early indicator that an IMP event, such as SFR, PDR, or CDR, will slip.

The solution for measuring SE tasks discretely has several elements. First, include significant accomplishments and accomplishment criteria for SE tasks and work products in the IMP. Next, show progress towards completing those SE work products in IMS and work packages. Typical SE work products include the system architecture (functional and physical), interface controls, specifications, trade studies, and test procedures.

For SE tasks such as defining and approving the product requirements, including derived requirements and allocated requirements, develop a requirements-based, time-phased budget that is based on the planned schedule for those requirements. Then base EV on the progress towards completing those requirements as recorded in the requirements data base. Typical examples of requirements status include defined, early validated, determined verification method, approved, allocated, and traced to a test procedure.

For work packages that result in SE work products that are technical measures (MOEs, MOPs, and TPMs), base EV on progress towards meeting the IMP criteria for their completion.

Plan Rework and Track it Discretely

Rework is frequently not adequately planned in the PMB and IMS. The rework can include rework of requirements analysis, design, and test tasks. Even if rework is belatedly budgeted from management reserve, it is often measured as level of effort, or if measured discretely, as a percent of the planned iterations. Neither technique reports progress towards developing or meeting the technical requirements.

The solution for better understanding and management of rework begins the proposal and the negotiated contract value. The program should verify realistic that rework assumptions and estimates are included in suppliers' proposals and negotiated values. The estimates should include productivity/quality measures such as rework percent and defect density.

The program should review the adequacy of budget and schedule for rework in the PMB. Rework should be planned in a separate planning package from the original task. When converted to a work package, it should be measured discretely based on technical maturity targets.

Establish interim milestones for rework with associated TPM planned values or quantified functionality based on meeting requirements. Then take interim EV based on net achieved technical performance. Make a negative adjustment to EV when necessary for accurate status reporting.

If rework is not in a separate work package and if EV had been taken for achieving a technical milestone, correct EV and the IMS when there is subsequent knowledge that the milestone completion criteria are now unmet. The milestone should be re-opened and a negative adjustment should be made to EV. Cumulative EV must reflect net technical progress.

New Contractual Requirements

New contractual requirements should be included in the Statement of Work (SOW) to communicate program needs. Some of the requirements are tantamount to tailoring several of the EVMS guidelines. The primary objective is to refocus management attention from the work scope to the product scope and to provide EV that truly reflects technical performance. Recommendations for acquisition reforms, including a revision to DFARS, are in a Defense AT&L article [5]. However, program offices can accomplish the same objectives by implementing the specific recommendations that follow.

1. For top down planning, make the IMP a contractual requirement and use a tailored CWBS DID.
2. Use tailored EVMS guidelines or specify EVM techniques in the SOW to:

a. Incorporate the product scope or technical baseline in the PMB.
b. Tie EV to technical performance.
c. Account for deferred functionality.
d. Track specified SE tasks discretely.
e. Plan rework and track it discretely.

IMP and SE Work Products

Require that an IMP be a contract deliverable. Start with the DoD IMP and IMS Preparation and Use Guide that is tailored to specify SE work products and accomplishment criteria. The IMP DID should be developed by the program SE organization.

An excellent source for specifying the SE tasks and work products is the Air Force Space and Missile Command Standard, SE Requirements and Products [6]. For example, it states that required SE products are: the SE accomplishments, accomplishment criteria, and narrative in the IMP; tasks in the IMS; and work packages in the EVMS, and such other specific plans (such as tradeoff plans) as may be needed to achieve the attributes required above.

CWBS DID, DI-MGMT-81334C

In practice, the CWBS does not include or point to the quantified technical or functional performance requirements that are in the specifications. Contractors will have to reference the functions at the CI level in the allocated functional baseline and product specifications in the product baseline. The contractual language is: The CWBS Dictionary for the appropriate CWBS elements must be updated to include or reference, at PDR, the functions allocated to one or more system CIs and, at CDR, the product specifications for each CI in the system.

Product Scope

With regard to Guideline 2.1a, authorized work, add contractual language to: "Include the work necessary to produce the product scope of the program, including rework (when applicable). The product scope is the technical baseline. It includes the features and functions that characterize a product or result."

Technical Performance

With regard to Guideline 2.2b, measure performance, add contractual language to specify that "All TPMs that have been identified at PDR shall be used to measure progress in appropriate work packages. Compare product and process metrics data against plans and schedule using trend analysis to determine technical areas requiring management attention."

Deferred Functionality

With regard to Guideline 2.5b, revisions, add contractual language to specify, "When work scope that is behind schedule is internally re-planned from the work package that is being closed to another open work package, the BCWR in the work package that is being closed shall be transferred to the first open period of the receiving work package The objective is to prevent arbitrary elimination of existing schedule variances. The time-phased estimate to complete of the receiving work package must be based on an analysis of remaining tasks in the IMS and projected resource plan."

Rework

With regard to Guideline 2.1a, authorized work, add contractual language to specify that: "The work scope includes rework. Rework includes corrective actions to hardware/software deficiencies, including deficiencies in the underlying requirements. Rework shall be planned, estimated, and included in the initial PMB. Rework shall be measured discretely and use technical performance goals to measure progress.

Conclusion

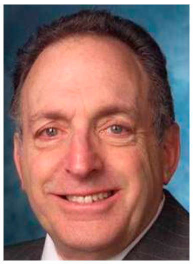
DoD has identified challenges to improve the usefulness and validity of EV information by integrating technical performance and systems engineering work products with EVM.

Implementation of the recommended acquisition management processes and new contractual requirements will provide the following benefits:

- Close the EVMS Quality Gap
- Insightful IBRs and technical reviews
- Valid contract performance reports
  - Objective technical/schedule status
  - Credible EAC
- Early detection of problems
  - Program performance
  - EV measurement and compliance

Incentives for suppliers to implement these process improvements can be implemented through the BBP 2.0 initiatives.

ABOUT THE AUTHOR



Paul J. Solomon, PMP, is co-author of the book, Performance-Based Earned Value.® He supported the B-2, Global Hawk, and F-35 programs at Northrop Grumman. He co-authored the EVMS Standard and received the DoD David Packard Excellence in Acquisition Award. He was a Visiting Scientist at the Software Engineering Institute and published "Using CMMI to Improve EVM." His web site, <www.PB-EV.com>, contains EVM best practices. He holds a BA and an MBA from Dartmouth College.

REFERENCES

1. Solomon, Paul J. "Practical Software Measurement, Performance-Based Earned Value." CROSSTALK Sept. 2001: 25-29 <http://www.crosstalkonline.org/storage/issue-archives/2001/200109/200109-Solomon.pdf>.
2. Solomon, Paul J. "Performance-Based Earned Value." CROSSTALK Aug. 2005: 25-26 <http://www.crosstalkonline.org/storage/issuarchives/2005/200508/200508-Solomon.pdf>.
3. Solomon, Paul J. "Practical Performance-Based Earned Value." CROSSTALK May 2006: 20-24 <http://www.crosstalkonline.org/storage/issue-archives/2006/200605/200605-Solomon.pdf>.
4. Solomon, Paul J. "Improving the Quality of EVM Information" JOURNAL OF SOFTWARE ENGINEERING Aug. 2011 <http://journal.thedacs.com/issue/58/195>.
5. Solomon, Paul J. "Path to EVM Acquisition Reform." DOD AT&L May 2011:25-27 <http://www.dau.mil/pubscats/ATL%20Docs/May-June11/Solomon.pdf>.
6. Air Force Space Command , Space and Missile Systems Center (SMC) Standard SMC-S-001, "SE Requirements and Products." July 2010. <http://www.everyspec.com/USAF/USAF+-+SMC/SMC-S-001_12JUL2010_21522/>

Core Estimating Concepts

William Roetzheim, Level 4 Ventures, Inc.

Abstract. Understanding the core estimating concepts will help you understand any of the currently available estimating tools and provide you with the framework you need when building new models for your particular problem domains. This article strips off the domain specific layers to get at the basic skeleton that underlies estimation in general.

Estimating Concepts

Most estimating articles, and tools, focus on domain specific models, benchmark data, and approaches. But for all labor related activities there are some generic concepts that underlie estimates for any of the types of work to be performed. These fundamental concepts apply whether you are using commercial parametric estimating tools or home-built Excel based models. User configurable cost estimating tools can be configured using these core concepts to support estimates for any labor driven work, or even for projects consisting of fundamentally different types of activities, even if the tool originally ships pre-initialized for a given domain.

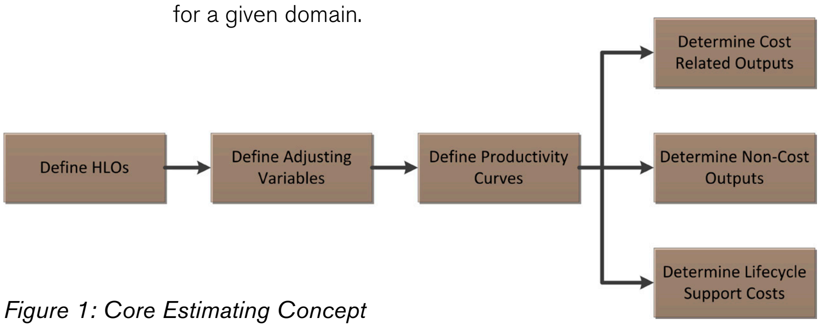

Define HLOs
Define Adjusting Variables
Define Productivity Curves
Determine Cost Related Outputs
Determine Non-Cost Outputs
Determine Lifecycle Support Costs

Figure 1: Core Estimating Concept

Figure 1: Core Estimating Concept, provides an overview of the estimating process at a sufficiently high level to ensure that it applies to estimating within any labor driven problem domain.

Step one in the process is to identify one or more High-level Objects (HLOs) that have a direct correlation with effort. The HLOs that are appropriate are domain specific, although there is sometimes an overlap. Examples of HLOs include yards of carpet to lay, reports to create, help desk calls to field, or claims to process. In activity-based costing, these would be the cost drivers. HLOs are often assigned a value based on their relative implementation difficulty, thereby allowing them to be totaled into a single numeric value. An example is function points, which are a total of the values for the function point HLOs.

HLOs may have an assigned complexity or other defining characteristics that cause an adjustment in effort (e.g., simple report versus average report). It is also typically necessary to have a technique for managing work that involves new development, modifications or extensions of existing components, or testing/validation only of existing components. Various formulas or simplifying assumptions may be used for this purpose. For example, in the case of reuse the original Constructive Cost Model (COCOMO) I model reduced the HLO size to:

$$\mathrm{HLO} = \mathrm{HLO} * (.4\mathrm{DM} + .3\mathrm{CM} + .3\mathrm{IT})$$

Where DM is the percent design modification (1% to 100%); CM is the percent code modification (1% to 100%); and IT is the percent integration and test effort (1% to 100%).

Step two is to define adjusting variables that impact either on productivity, or on economies (or diseconomies) of scale. The productivity variables tend to be things like the characteristics of the labor who will be performing the work or the tools they will be working with; characteristics of the products to be created (e.g., quality tolerance) or the project used to create them; and characteristics of the environment in which the work will be performed. The variables that impact on economies or diseconomies of scale are typically things that drive the necessity for communication/coordination, and the efficiency of those activities. These adjusting variables are important both to improve the accuracy of any given estimate, and also to normalize data to support benchmarking across companies or between application areas.

Step three involves defining productivity curves. These are curves that allow a conversion between adjusted HLO sizing counts and resultant effort. They are typically curves (versus lines) because of the economies or diseconomies of scale that are present. Curves may be determined empirically or approximated using industry standard data for similar domains. Curves may also be adjusted based on the degree to which the project is rushed. In any event, procedures are put in place to collect the necessary data to support periodic adjustment of the curves to match observed results, a process called calibration.

The outputs of the process are driven by the needs of the organization. These outputs can be broken down into three major categories:

1. Cost (or effort, which is equivalent for this purpose): In addition to the obvious total value, most organizations are interested in some form of breakdown. Typical breakdowns include breakdowns by organizational unit for budgetary or resource planning purposes; breakdowns by type of money from a generally accepted accounting principles perspective (e.g., opex versus capex); or breakdown by work breakdown structure elements in a project plan. These outputs will also typically include labor needed over time, broken down by labor category. These outputs are generated using a top down allocation.

2. Non-cost Outputs: Non-cost outputs are quantitative predictions of either intermediate work product size, or non-cost deliverable components. Examples include the number of test cases (perhaps broken down by type), the engineering documents created with page counts, the number of use-case scenarios to be created, or the estimated help desk calls broken down by category. These outputs are typically created using curves similar to the productivity curves, operating either on the HLOs or on the total project effort.

3. Lifecycle Costs: If the estimate is for a product to be created, delivered, and accepted then the cost and non-cost items above would typically cover the period through acceptance. In most cases there would then be an on-going cost to support and maintain the delivered product throughout its lifecycle. These support costs are relatively predictable both in terms of the support activities that are required and the curves that define the effort involved. For many of them, the effort will be high immediately following acceptance, drop off over the course of one to three years to a low plateau, then climb again as the product nears the end of its design life.

Understanding these basic concepts, it is clear that for a given system there may be many different estimates that need to be prepared and combined. Each aspect of the work that involves different HLOs, different adjusting variables, or different productivity curves is really a different model. But all of the models rest within a consistent framework, and in fact, can run within the same tool. There is another dimension of the estimate we need to consider: project lifecycle, or time.

For most projects, it is impossible to completely and accurately define the end product to be delivered. In fact, I would argue that the only way to completely avoid uncertainty in the end product is to have an exact model of the desired results before you start, and it is unusual to have such a model available. In fact, most of the effort spent on projects is on a progressive elaboration of the baseline description of what is to be ultimately delivered. As shown in Figure 2: Progressively Elaborated Baseline," the baseline of what will ultimately be delivered is progressively elaborated throughout the life of the project. Using software as an example, the requirement specification elaborates the functional baseline; the design elaborates the requirement specification; and the code elaborates the design.

As a project moves through this process of progressive elaboration, the estimation models also progress forward (see Figure 3: Estimating Lifecycle). At the most obvious level, as you understand the problem more you can more accurately decompose the work to be performed and prepare an estimate. However, there is another phenomenon at work. The actual estimation model components will change as you move through the process. For example, the HLOs that are used to define the product(s) will change, becoming more and more granular as you move forward. At the high-level estimate stage you might think in terms of a new screen including supporting back-end processing and middleware communication components; at the scope estimate you might be looking at a screen, a table, and a new service; and at the validation estimate stage you might be talking in terms of stored procedures to be written. They are all different perspectives of the same functionality that will ultimately be delivered, but with different levels of granularity. However, the core components of Figure 1 are the same for all of these estimates.

Not only are better estimates possible as you move through the project life, but the primary reason for doing the estimate will change over time. Take a look at Figure 4: Estimating Purposes. Early lifecycle high-level estimates are often used for demand management. Projects are examined for feasibility and selected based on ROI or other financial measures that require estimates to perform the calculations. Scarce resources are allocated to support planned projects based on these demand estimates. One characteristic of high level estimates is that a significant percentage of the projects that are estimated (as high as 90% in some cases) are never started. Once a project is at least partially funded and the requirements are better understood and defined (i.e., the baseline has been progressively elaborated one level), then a scope level estimate can be prepared. In many organizations, this is called a "commit" estimate because this will be the estimate used as a basis for measuring project success going forward. The scope level estimate defines the project baseline estimate. Changes in scope are then estimated and, if approved, those estimates are used to modify the baseline. When the project is complete, an as-built sizing is performed to update the organization historical database and for calibration.

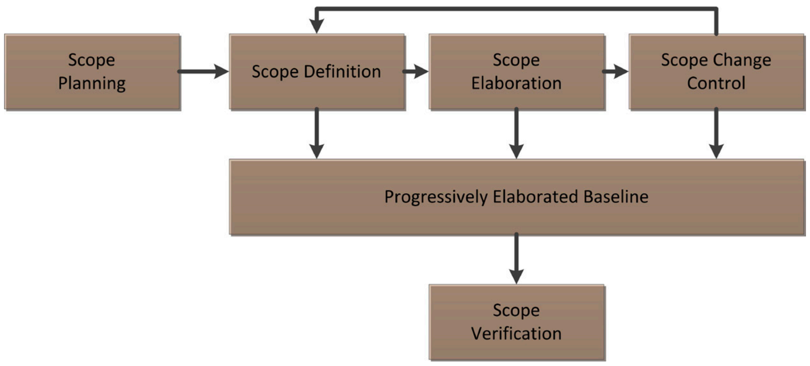


Figure 2: Progressively Elaborated Baseline

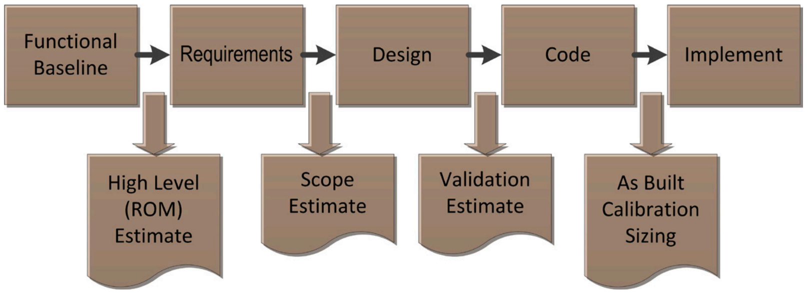


Figure 3: Estimating Lifecycle

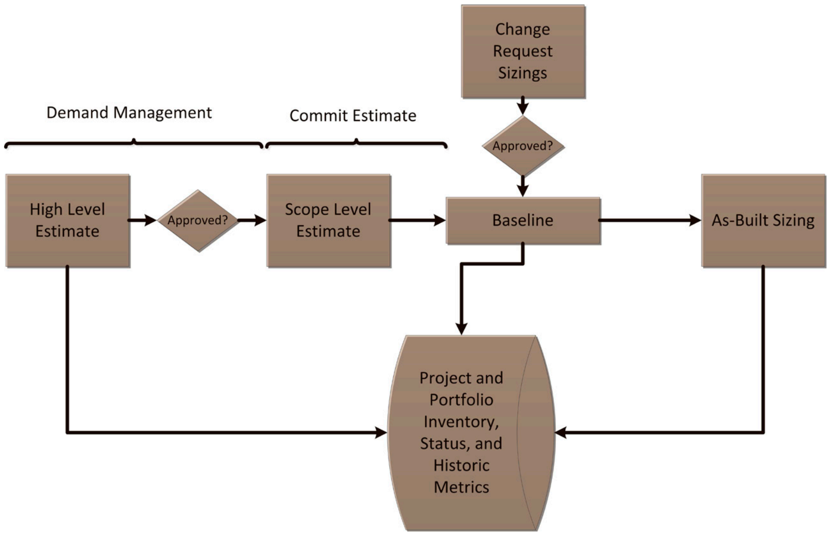


Figure 4: Estimating Purposes

One final core concept of cost estimating is worth discussing: The difference between an estimate and a budget. An estimate is defined as the most likely outcome of a probabilistic event, taking into consideration everything that is currently known about the project. However, the estimate does not include risk, an important component of the project budget. As shown in Figure 5: Estimating versus Budgeting, the estimate defines a starting baseline. Your risk management process (shown at the top of the figure) will then determine the necessary funds for contingency funds and risk response funds. Risk response funds are planned expenditures designed to reduce negative risk or enhance positive risk (opportunities). Risk Response Funds will always be a cost to the project. Contingency funds are monies set aside to deal with risks that are known but uncertain. Generally, these will be a net cost to a project, although in some situations where risk management has identified some significant positive risks, they may actually reduce the project budget. Finally, the organization will normally want to include a management reserve to allow for unknown-unknowns, or risks that are not discovered until later in the project life.

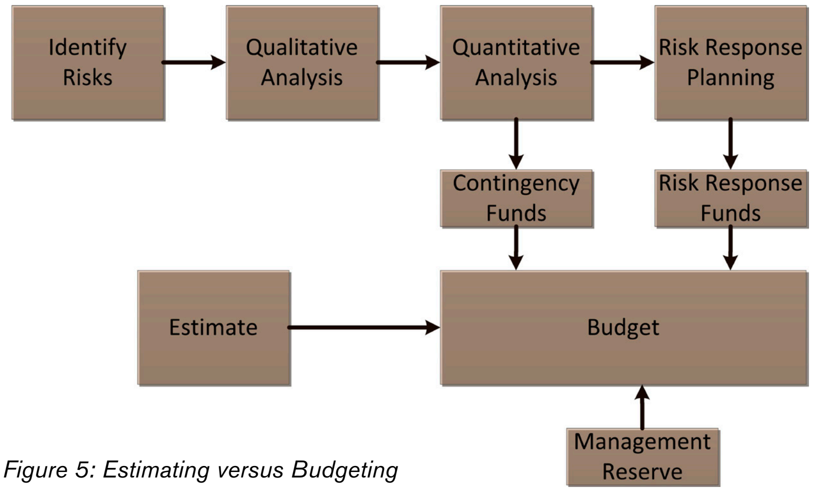


Figure 5: Estimating versus Budgeting

Putting it All Together

Let us take a look at how all of this fits, starting with a slide prepared by the Naval Center for Cost Analysis and presented by Mr. Bryan Flynn at the 43nd Annual DODCAS [1]. As shown in Figure 6: DON Cost Estimating Standard, the DON standard approach aligns well with the approach just described. We will look at it step-by-step, using some examples to explain the process.

Step 1: Establish Needs With Customer

While not directly addressed in this article, this project initiation step is actually the most critical, yet the most often overlooked. I often say that good software cost analysis is 90% stakeholder management, and 10% math. And the key to stakeholder management is understanding the needs of the stakeholders.

Step 2: Establish a Program Baseline

Here we are reviewing the business requirements and acquisition strategy (perhaps captured in a cost analysis requirements description), identifying cost drivers or HLOs of this article; and identifying risk areas (the start of risk analysis). For example, in

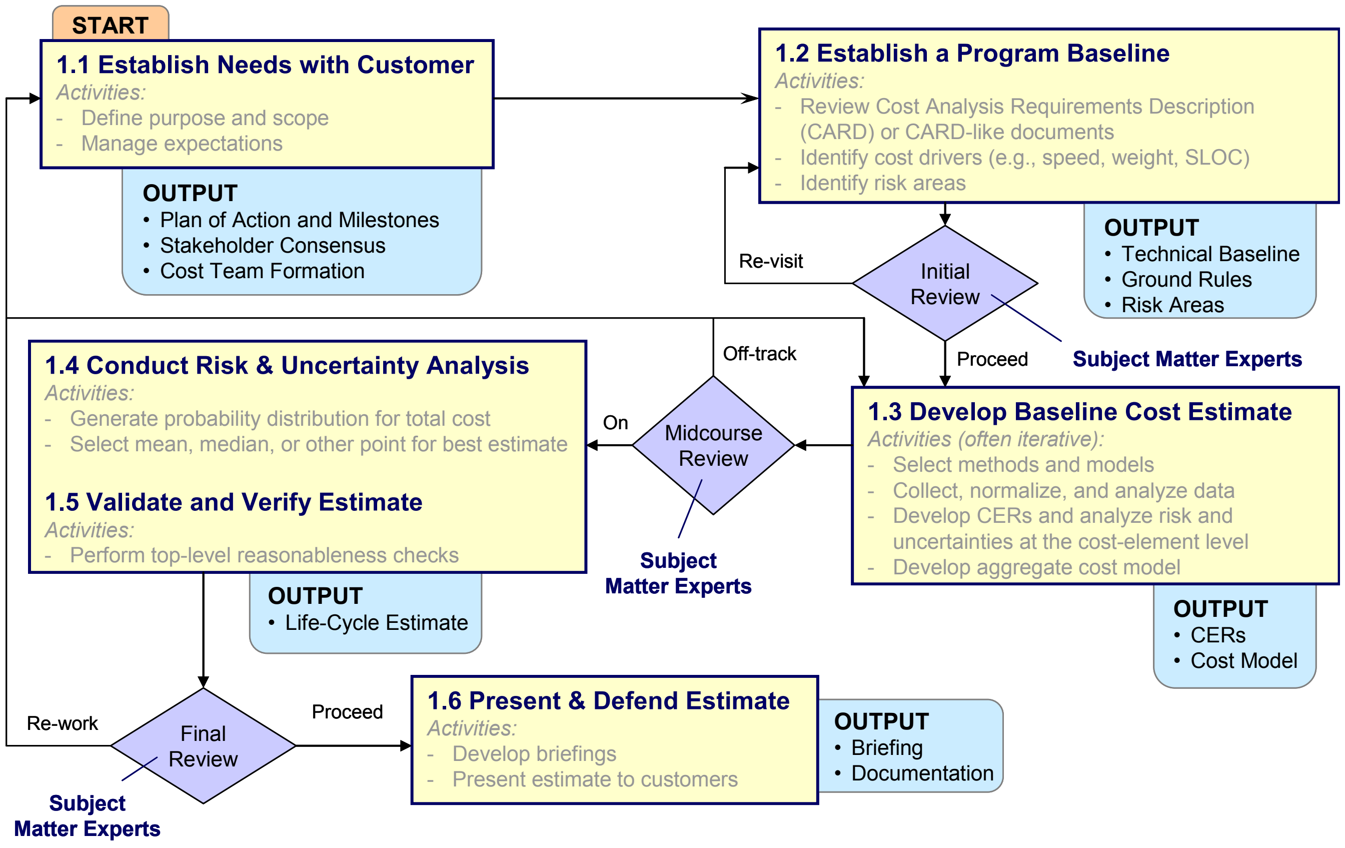

START
1.1 Establish Needs with Customer
Activities:
- Define purpose and scope
- Manage expectations
OUTPUT
• Plan of Action and Milestones
• Stakeholder Consensus
• Cost Team Formation
1.2 Establish a Program Baseline
Activities:
- Review Cost Analysis Requirements Description (CARD) or CARD-like documents
- Identify cost drivers (e.g., speed, weight, SLOC)
- Identify risk areas
Re-visit
Initial Review
OUTPUT
• Technical Baseline
• Ground Rules
• Risk Areas
Proceed
Subject Matter Experts
Off-track
1.4 Conduct Risk & Uncertainty Analysis
Activities:
- Generate probability distribution for total cost
- Select mean, median, or other point for best estimate
1.5 Validate and Verify Estimate
Activities:
- Perform top-level reasonableness checks
On
Midcourse Review
Subject Matter Experts
1.3 Develop Baseline Cost Estimate
Activities (often iterative):
- Select methods and models
- Collect, normalize, and analyze data
- Develop CERs and analyze risk and uncertainties at the cost-element level
- Develop aggregate cost model
OUTPUT
• Life-Cycle Estimate
OUTPUT
• CERs
• Cost Model
Re-work
Final Review
Proceed
1.6 Present & Defend Estimate
Activities:
- Develop briefings
- Present estimate to customers
OUTPUT
• Briefing
• Documentation
Subject Matter Experts

Figure 6: DON Cost Estimating Standard

conducting an analysis of a large DoD ERP implementation, we looked at the available requirement document and determined that the most logical HLOs would be Reports, Interfaces, Conversions, Enhancements, and Workflows. We not only collected together the count of each, but also assigned a complexity value to each (very low, low, average, high, very high) and differentiated between those that were new versus those that were modified. For the modified objects, we estimated the extent of the modification (low, medium, high). In this case, we had historic information that allowed us to estimate both the relative effort for each type of HLO, plus the spread between very low to very high complexity for each type of HLO.

Step 3: Develop Baseline Cost Estimate

The methods and models that are mentioned here are our productivity curves. What we want is models or methods that will allow us to convert between HLOs and effort. Or more broadly, we might say that we are looking at cost curves to convert between HLOs and cost, assuming that we can develop models encompassing non-labor cost driver equations.

The activity of normalizing data discussed here actually happens at multiple points in the process. First, HLO types are normalized relative to each other through some form of relative weighting in terms of effort (or cost). Second, the cost curves are normalized through project specific adjustments, our adjusting variables.

The cost estimating relationships from the figure are at the heart of the allocation process used to generate our cost and non-cost related outputs.

For the ERP estimate that we are using as our example, we first want to estimate the total effort. For this we start with a suitable productivity model based on the lifecycle being used and the historic data set used for the analysis. The resultant equation is of the form:

$$\text{Effort} = \alpha * \text{Size}^{\beta}$$

Where α and β are the constants of the model and Size is the normalized total of the HLO values. We then look at project and organizational specific adjustments to α and β. What we are really interested in here are differences between this project/organization and the historic projects that we used. A couple of good sources to look for potential changes and their likely impact on the variables are the COCOMO II environmental variables and the IFPUG General System Characteristics, although those are by no means the only valid sources.

Step 4: Conduct Risk and Uncertainty Analysis

The activities described here deal with probabilistic variances in the cost estimate based on uncertainty in the estimation process itself. While these are certainly one source of risk, they are not the only source of risk. It is probably more generically correct to follow the PM-BOK approach described in this article, in which an allowance is added to the estimate to allow for risk mitigation activities, risk contingency funds based on the expected value of the risk factors at work, plus some form of management reserve based on the risk tolerance of the organization and the nature of the project.

Step 5: Validate and Verify Estimate

A key mistake many novice estimators make is to bury their head in their spreadsheets and end up with results that go against common sense. In the Naval Aviation field, we would have talked about the necessity for a pilot to, "Get their head out of the cockpit."

Of course, just because an estimate goes against common sense does not mean it is wrong. I have seen many situations where the models were right and common sense was wrong. But it does mean that you should take another look to make sure you are not making an error of some kind.

And of course, the validation of an estimate may go beyond a gut check. It is often possible (and useful) to attack the problem using two or more different approaches and to then see if the results converge. For example, you might compare a parametric estimate with a bottom up estimate, or you might prepare two estimates using different HLOs as the sizing input. An estimate by analogy is often a good validation approach. This basically involves finding one or more other projects that is similar to this project, adjusting for any differences, and comparing the adjusted historical values to the current estimate. Another approach that is sometimes used is to compare the results from two or more commercial estimating tools.

Step 6: Present and Defend Estimate

Yes, of course this is necessary. But what is also necessary is the step of updating the estimate as additional information becomes available throughout the life of the project.

Conclusions

My goal in writing this article was to define estimating in terms of the fundamental concepts that would pertain no-matter what type of estimate you were creating and no matter what tool you were employing. This understanding of the big picture is useful both in understanding how estimating models and tools work, and also in developing new models or tools for domains where existing models do not exist. ❖

ABOUT THE AUTHOR

William Roetzheim is founder and CEO of Level 4 Ventures, Inc. He has written 27 published books, more than 100 articles, and three columns. He has been a frequent lecturer and instructor at multiple technology conferences and two California universities. Mr. Roetzheim has an MBA, is an IFPUG certified function point counter, is a Certified Cost Estimation Analyst (CCEA), and has both a PMP and RMP designation by the Project Management Institute.

REFERENCES

1. Flynn, Bryan: "DoD/DON Acquisition Instructions and DON Cost Estimating Standard," presented at the Department of Defense Cost Analysis Symposium, The Lodge at Williamsburg, Virginia, 19 February 2010.

Statistical Tune-Up of the Peer Review Engine to Reduce Escapes

Tom Lienhard, Raytheon Missile Systems

Abstract. Peer reviews are a cornerstone to the product development process. They are performed to discover defects early in the lifecycle when they are less costly to fix. The theory is to detect the defects as close to the injection point as possible reducing the cost and schedule impact. Like most, if not all companies, peer reviews were performed and data collected allowing characterization of those reviews. Data collected across the organization showed that more than 30% of the engineering effort was consumed by reworking products already deemed fit for purpose. That meant for every three engineers a fourth was hired just to rework the defects. This was unacceptable!

The major contributor to this rework was defects that escaped or "leaked" from one development phase to a later phase. In other words, the peer reviews were not detecting defects in the phase during which they were injected. Defect leakage is calculated as a percentage, by summing the defects attributable to a development phase that are detected in later phases divided by the total number of defects attributable to that phase. Defect leakage leads to cost and budget over-runs due to excessive

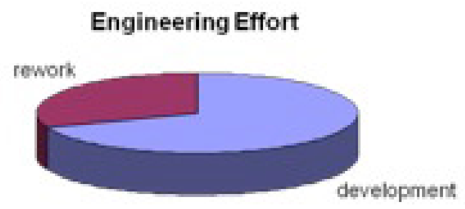


Figure 1

rework. For some development phases, defect leakage was as high as 75%. By investigating the types of defects that go undetected during the various development phases, corrections can be introduced into the processes to help minimize defect leakage and improve cost and schedule performance. An organizational goal was then set at no more than 20% defect leakage.

To perform this investigation and propose improvements, a suite of Six Sigma tools were used to statistically tune-up the peer review process. These tools included Thought Process map, Process Map, Failure Mode and Effect Analysis, Product Scorecard, Statistical Characterization of Data, and a Design of Experiments.

Having been an engineer and process professional for more than 20 years, I knew (or thought I knew) what influenced the peer review process and what needed to be changed in the process. But when we began the process, I kept an open mind and used Six Sigma tools to characterize and optimize the peer review process.

The Thought Process Map was needed to scope the project, keep the project on track, identify barriers, and document results. It was useful to organize progress and eliminate scope-creep.

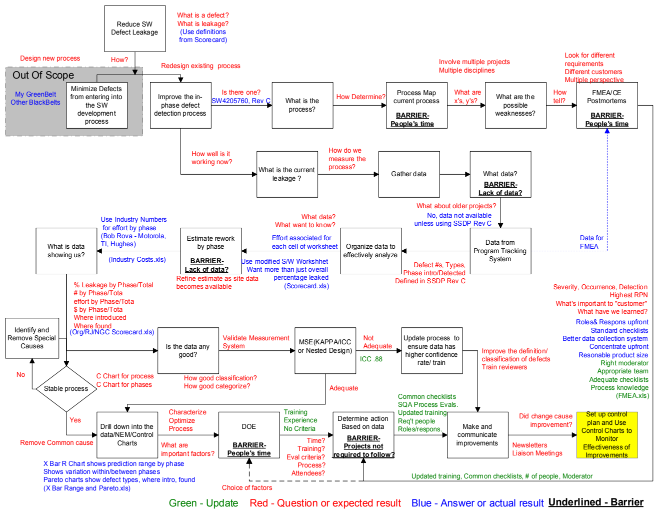
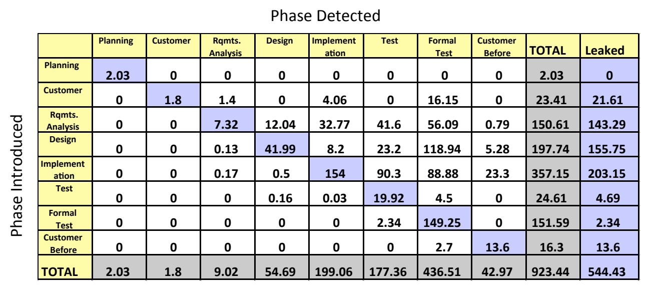

Reduce SW Defect Leakage
What is a defect?
What is leakage?
(Use definitions from Scorecard)
Design new process
How?
Redesign existing process
Out Of Scope
My GreenBelt
Other BlackBelts
Minimize Defects from entering into the SW development process
Improve the in-phase defect detection process
Is there one?
SW4205760, Rev C
What is the process?
How Determine?
Process Map current process
BARRIER-People's time
Involve multiple projects
Multiple disciplines
What are x's, y's?
What are the possible weaknesses?
How tell?
Look for different requirements
Different customers
Multiple perspective
FMEA/CE Postmortems
BARRIER-People's time
How well is it working now?
What is the current leakage ?
How do we measure the process?
Gather data
What data?
BARRIER-Lack of data?
What about older projects?
No, data not available unless using SSDP Rev C
Data from Program Tracking System
Data for FMEA
Defect #s, Types, Phase intro/Detected Defined in SSDP Rev C
Organize data to effectively analyze
What data?
What want to know?
Effort associated for each cell of worksheet
Use modified S/W Workshhet
Want more than just overall percentage leaked
(Scorecard.xls)
Estimate rework by phase
BARRIER-Lack of data?
Refine estimate as site data becomes available
Use Industry Numbers for effort by phase
(Bob Rova - Motorola, TI, Hughes)
(Industry Costs.xls)
What is data showing us?
% Leakage by Phase/Total
# by Phase/Tota
effort by Phase/Tota
$ by Phase/Tota
Where introduced
Where found
(Org/RJ/NGC Scorecard.xls)
Identify and Remove Special Causes
No
Stable process
C Chart for process
C Chart for phases
Is the data any good?
How good classification?
How good categorize?
Validate Measurement System
MSE(KAPPA/ICC or Nested Design)
Not Adequate
ICC .88
Adequate
Update process to ensure data has higher confidence rate/ train
Improve the definition/classification of defects
Train reviewers
Severity, Occurrence, Detection
Highest RPN
What's important to "customer"
What have we learned?
Roles& Respons upfront
Standard checklists
Better data collection system
Concentrate upfront
Resonable product size
Right moderator
Appropriate team
Adequate checklists
Process knowledge
(FMEA.xls)
Yes
Remove Common cause
Drill down into the data/NEM/Control Charts
Characterize
Optimize
Process
What are important factors?
X Bar R Chart shows prediction range by phase
Shows variation within/between phases
Pareto charts show defect types, where intro, found
(X Bar Range and Pareto.xls)
DOE
BARRIER-People's time
Choice of factors
Training
Experience
No Criteria
Time?
Training?
Eval criteria?
Process?
Attendees?
Determine action Based on data
BARRIER-Projects not required to follow?
Common checklists
SQA Process Evals.
Updated training
Req't people
Roles/respons.
Make and communicate improvements
Did change cause improvement?
Newsletters
Liaison Meetings
Set up control plan and Use Control Charts to Monitor Effectiveness of Improvements
Updated training, Common checklists, # of people, Moderator
Green - Update
Red - Question or expected result
Blue - Answer or actual result
Underlined - Barrier

Figure 2

The Process Map was used to "walk the process" as it is implemented—not as it was defined in the command media. Inputs, outputs, and resources were identified. Resources were categorized as critical, noise, standard operating procedure and controllable. The Process Map was extremely useful because it quickly highlighted duplicate activities, where implementation deviated from the documented process, and was used as an input to the Failure Mode and Effect Analysis (FMEA) and Design of Experiments (DOE).

The FMEA leveraged the process steps from the Process Map to identify potential failure modes with each process step, the effect of the failure, the cause of the failure, and any current detection mechanism. A numerical value was placed on each of these attributes and a cumulative Risk Priority Number (RPN) was assigned to each potential failure. The highest RPNs were the potential failures that needed to be mitigated or eliminated first and would eventually become the factors for the DOE.

The Product Scorecard contained all of the quantifiable data relating to the peer reviews. It showed the number of defects introduced and detected by phase, both in raw numbers, percentage, and by effort. Using Pareto Charts, it was easy to determine where defects entered the process, where defects were found by the process, and even which phases had the most impact (rework) to the bottom line. Surprisingly, 58% of the total detects were found in test, well after the product is deemed "done". Additionally, three phases accounted for greater than 92% of rework due to defects.



Phase Detected

Phase Introduced

	Planning	Customer	Rqmts. Analysis	Design	Implementation	Test	Formal Test	Customer Before	TOTAL	Leaked
Planning	2.03	0	0	0	0	0	0	0	2.03	0
Customer	0	1.8	1.4	0	4.06	0	16.15	0	23.41	21.61
Rqmts. Analysis	0	0	7.32	12.04	32.77	41.6	56.09	0.79	150.61	143.29
Design	0	0	0.13	41.99	8.2	23.2	118.94	5.28	197.74	155.75
Implementation	0	0	0.17	0.5	154	90.3	88.88	23.3	357.15	203.15
Test	0	0	0	0.16	0.03	19.92	4.5	0	24.61	4.69
Formal Test	0	0	0	0	0	2.34	149.25	0	151.59	2.34
Customer Before	0	0	0	0	0	0	2.7	13.6	16.3	13.6
TOTAL	2.03	1.8	9.02	54.69	199.06	177.36	436.51	42.97	923.44	544.43

Figure 3

An improvement goal was set by the organization. The immediate goal was set around finding the defects earlier in the lifecycle rather than trying to reduce the number of defects. If the process could be improved to find the defects just one phase earlier in the lifecycle, the result would be many hundreds of thousands of dollars to the bottom line!

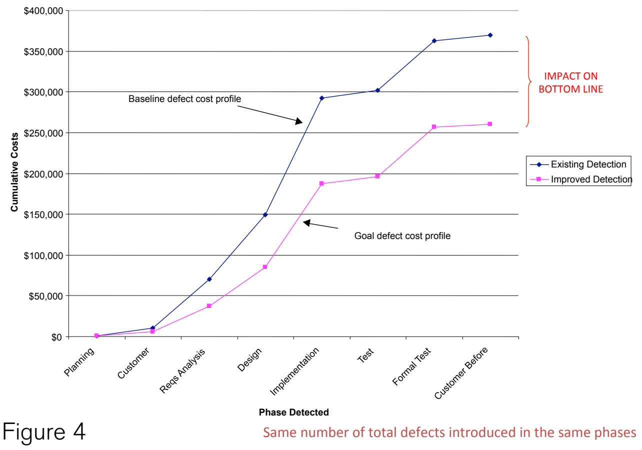


Figure 4

The data from the Product Scorecard was plotted to create a distributional characteristic of the process capability. Visually, this highlighted the lifecycle phases that were well below our goal of finding 80% of defects in phase, as seen in the figure below.

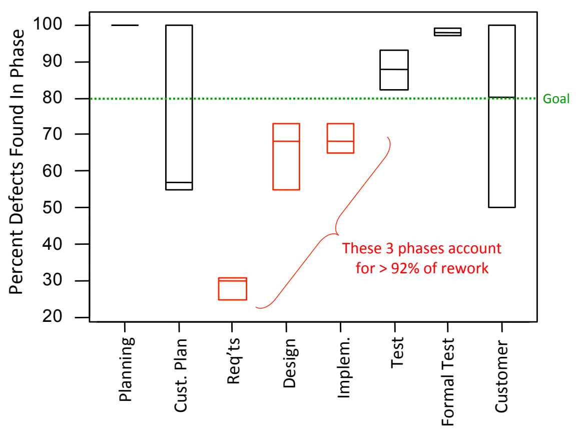


Figure 5

Going into this project, my belief was that a program could be identified that was conducting peer reviews effectively across the entire lifecycle and that program's process could be replicated across the organization. The Control Chart showed something quite different. All the programs were conducting peer reviews consistently, but the variation between lifecycle phases ranged widely. When the data was rationally subgrouped by phase, the data became stable (predictable) within the subgroups, but there was extensive variation between the subgroups. This meant the variation came from the lifecycle phases not the programs. It would not be as simple as finding the program that conducted effective peer reviews and replicating its process across the organization.

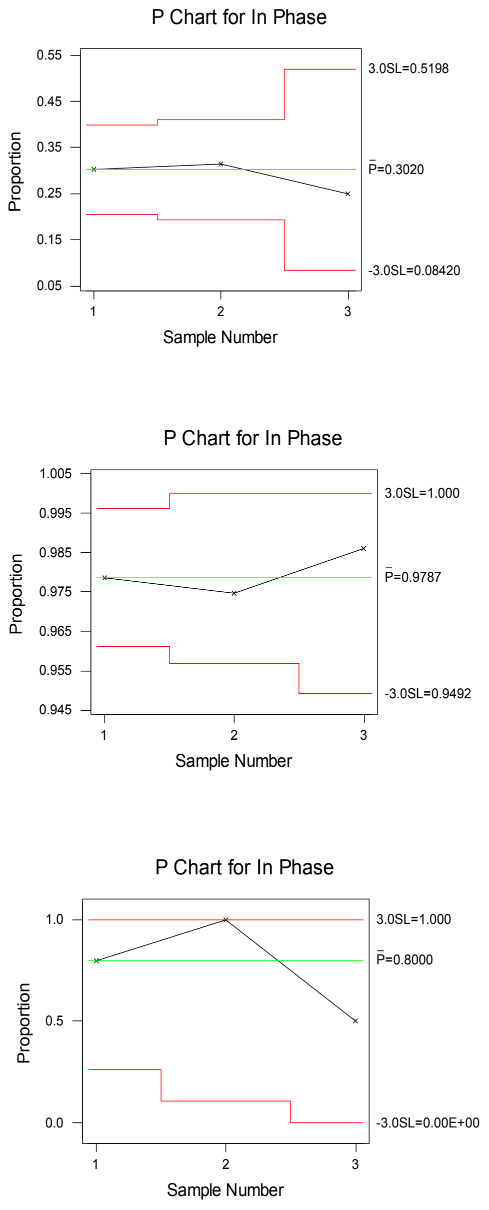
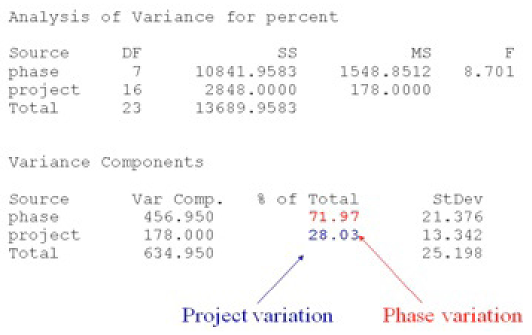


Figure 6

The Analysis of Variation confirmed that 72% of the process variation was between the subgroups (lifecycle phases) and only 28% was within the subgroup (programs). Since the data was only a sample of the population, Confidence Intervals were conducted to find out the true range of the population. This quickly showed that for the Requirements Phase, the best the process was capable of achieving was detecting 37% of defects in phase. In fact, if no action was taken it was 95% certain that the Requirements Phase will find between 21% - 37% of defects in phase, the Design Phase will find between 42% - 88% of defects in phase, the Implementation Phase will find 59% - 78% of defects in phase. This helped focus where to concentrate the improvement resources.



```
Analysis of Variance for percent

Source      DF          SS          MS        F        P
phase        7   10841.9583   1548.8512   8.701    0.000
project     16    2848.0000    178.0000
Total       23   13689.9583

Variance Components

Source       Var Comp.   % of Total      StDev
phase          456.950        71.97     21.376
project        178.000        28.03     13.342
Total          634.950                  25.198
```

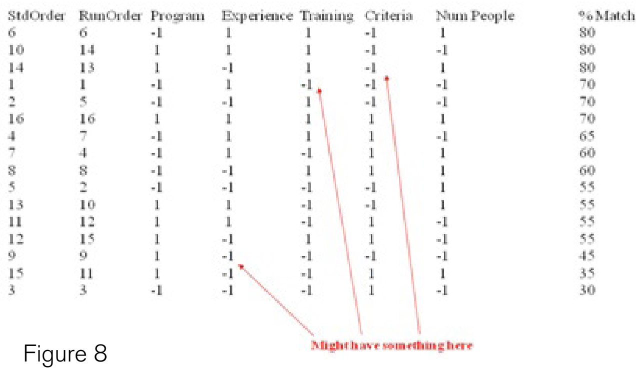
Project variation
Phase variation

Figure 7

Remember the high RPNs from the FMEA? These were used as the factors in a DOE. There were four factors (experience, training, review criteria, and number of reviewers). The response variable for the DOE was the percentage of defects found in a peer review. There were 16 runs, which made it a half-factorial DOE.

There were some limitations with this DOE. The products reviewed were different for each run; there were restrictions on randomization; and by the latter runs it was hard to find a peer review team that fulfilled the factor levels. For example, once somebody was trained they could not be untrained.

When analyzing data, always think golf (PGA = practical, graphical, and analytical). Practical analysis looked at the result of each for anything of interest. It was not until then the runs were sorted by response did any trends appear. The highest five runs all had no criteria, the lowest four consisted of inexperienced team members and six of the top seven were trained teams.



StdOrder	RunOrder	Program	Experience	Training	Criteria	Num People	% Match
6	6	-1	1	1	-1	1	80
10	14	1	1	1	-1	-1	80
14	13	1	-1	1	-1	1	80
1	1	-1	1	-1	-1	-1	70
2	5	-1	-1	1	-1	-1	70
16	16	1	1	1	1	1	70
4	7	-1	1	1	1	-1	65
7	4	-1	1	-1	1	1	60
8	8	-1	-1	1	1	1	60
5	2	-1	-1	-1	-1	1	55
13	10	1	1	-1	-1	1	55
11	12	1	1	-1	1	-1	55
12	15	1	-1	1	1	-1	55
9	9	1	-1	-1	-1	-1	45
15	11	1	-1	-1	1	1	35
3	3	-1	-1	-1	1	-1	30

Might have something here

Figure 8

Graphical analysis included a normal probability plot and a Pareto chart of the main effects, two-way and three-way effects. This clearly showed that training, criteria, and experience were the influential factors.

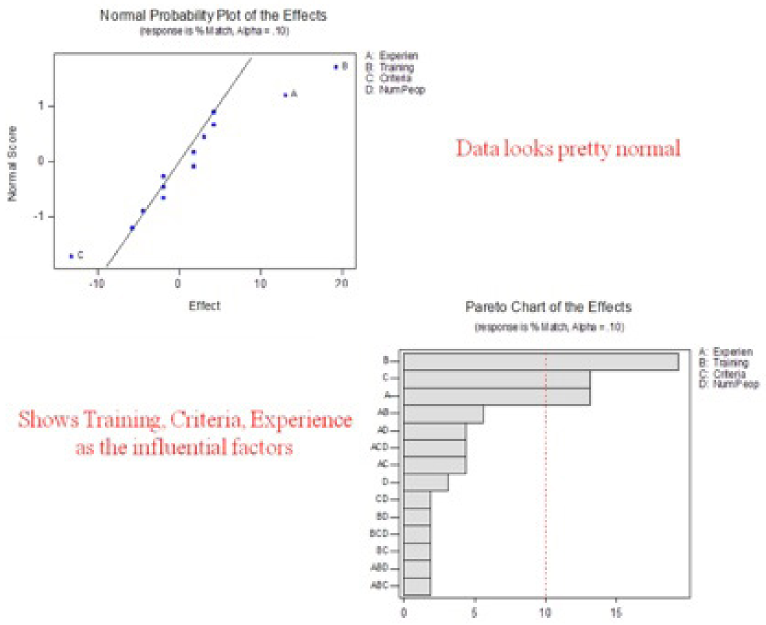
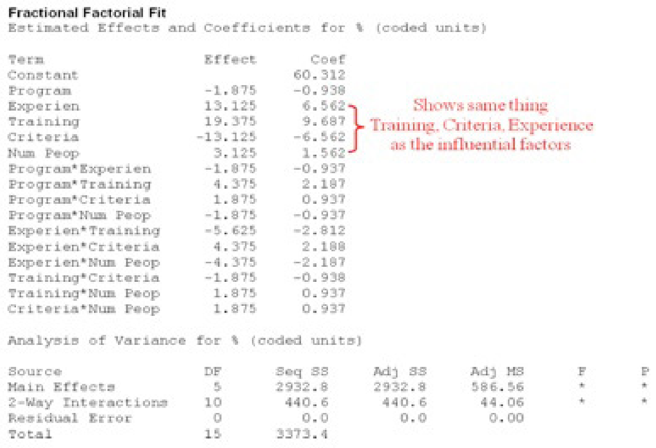


Figure 9

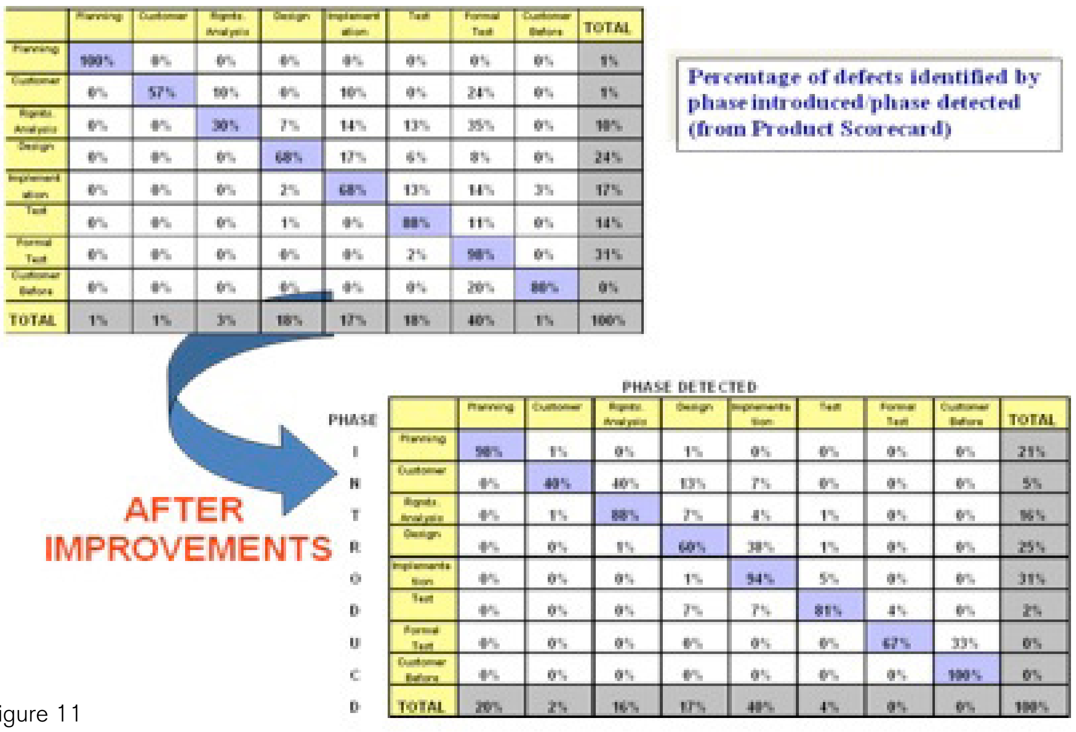
Analytical analysis not only showed the same influential factors but also quantified the effect and indicated whether to set the factor high or low. Training was the most influential, followed closely by experience. The process was relatively robust with respect to the language and number of people. If peer reviews are just as effective with half the people, this alone could have a big savings to the bottom line. The eye-opener here was that the peer review process was more effective without criteria. This went against intuition, but was based on data.



```
Fractional Factorial Fit
Estimated Effects and Coefficients for % (coded units)

Term                      Effect       Coef
Constant                              60.312
Program                   -1.875      -0.938
Experien                  13.125       6.562
Training                  19.375       9.687
Criteria                 -13.125      -6.562
Num Peop                   3.125       1.562
Program*Experien          -1.875      -0.937
Program*Training           4.375       2.187
Program*Criteria           1.875       0.937
Program*Num Peop          -1.875      -0.937
Experien*Training         -5.625      -2.812
Experien*Criteria          4.375       2.188
Experien*Num Peop         -4.375      -2.187
Training*Criteria         -1.875      -0.938
Training*Num Peop          1.875       0.937
Criteria*Num Peop          1.875       0.937

Analysis of Variance for % (coded units)

Source                    DF      Seq SS      Adj SS      Adj MS       F        P
Main Effects               5      2932.8      2932.8      586.56       *        *
2-Way Interactions        10       440.6       440.6       44.06       *        *
Residual Error             0         0.0         0.0        0.00
Total                     15      3373.4
```

Shows same thing Training, Criteria, Experience as the influential factors

Figure 10


Percentage of defects identified by phase introduced/phase detected (from Product Scorecard)
AFTER IMPROVEMENTS
PHASE DETECTED
PHASE INTRODUCED

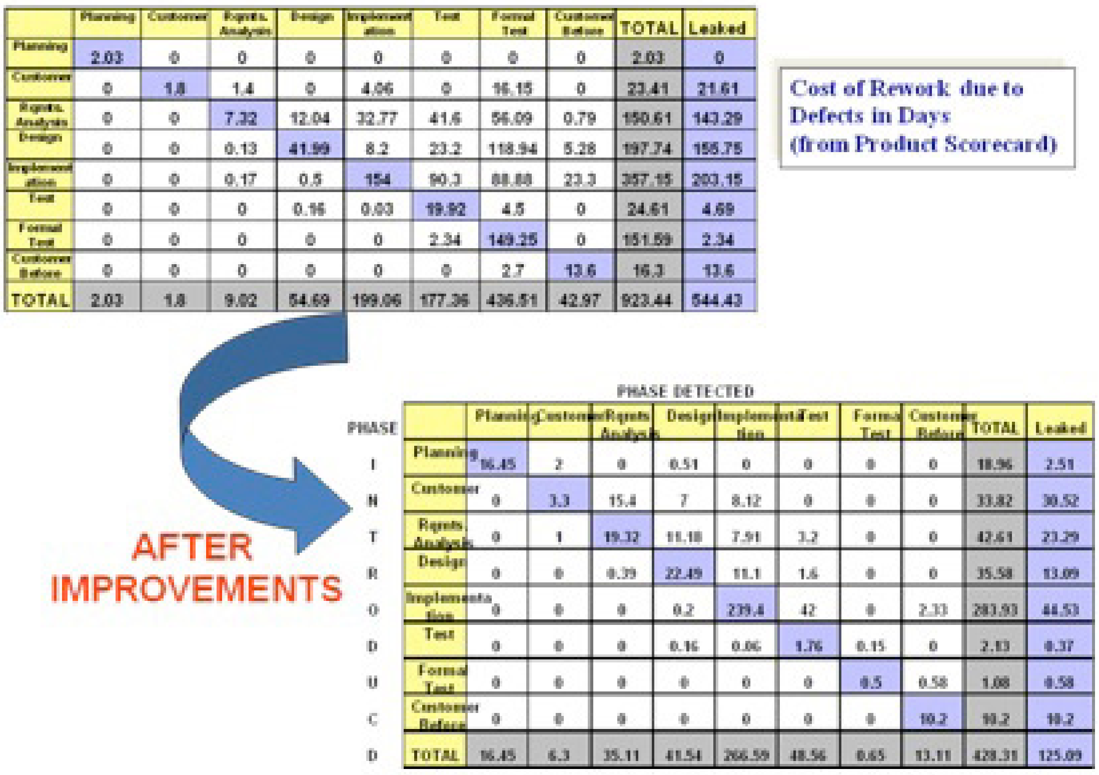
	Planning	Customer	Rqmts. Analysis	Design	Implementation	Test	Formal Test	Customer Before	TOTAL
Planning	100%	0%	0%	0%	0%	0%	0%	0%	1%
Customer	0%	57%	10%	0%	10%	0%	24%	0%	1%
Rqmts. Analysis	0%	0%	30%	7%	14%	13%	35%	0%	10%
Design	0%	0%	0%	68%	17%	6%	8%	0%	24%
Implementation	0%	0%	0%	2%	68%	13%	14%	3%	17%
Test	0%	0%	0%	1%	0%	88%	11%	0%	14%
Formal Test	0%	0%	0%	0%	0%	2%	98%	0%	31%
Customer Before	0%	0%	0%	0%	0%	0%	20%	80%	0%
TOTAL	1%	1%	3%	18%	17%	18%	40%	1%	100%

AFTER IMPROVEMENTS

PHASE DETECTED

PHASE INTRODUCED	Planning	Customer	Rqmts. Analysis	Design	Implementation	Test	Formal Test	Customer Before	TOTAL
Planning	98%	1%	0%	1%	0%	0%	0%	0%	21%
Customer	0%	40%	40%	13%	7%	0%	0%	0%	5%
Rqmts. Analysis	0%	1%	88%	7%	4%	1%	0%	0%	16%
Design	0%	0%	1%	60%	38%	1%	0%	0%	25%
Implementation	0%	0%	0%	1%	94%	5%	0%	0%	31%
Test	0%	0%	0%	7%	7%	81%	4%	0%	2%
Formal Test	0%	0%	0%	0%	0%	0%	67%	33%	0%
Customer Before	0%	0%	0%	0%	0%	0%	0%	100%	0%
TOTAL	20%	2%	16%	17%	40%	4%	0%	0%	100%

Figure 11


Cost of Rework due to Defects in Days (from Product Scorecard)
AFTER IMPROVEMENTS
PHASE DETECTED
PHASE INTRODUCED

	Planning	Customer	Rqmts. Analysis	Design	Implementation	Test	Formal Test	Customer Before	TOTAL	Leaked
Planning	2.03	0	0	0	0	0	0	0	2.03	0
Customer	0	1.8	1.4	0	4.06	0	16.15	0	23.41	21.61
Rqmts. Analysis	0	0	7.32	12.04	32.77	41.6	56.09	0.79	150.61	143.29
Design	0	0	0.13	41.99	8.2	23.2	118.94	5.28	197.74	155.75
Implementation	0	0	0.17	0.5	154	90.3	88.88	23.3	357.15	203.15
Test	0	0	0	0.16	0.03	19.92	4.5	0	24.61	4.69
Formal Test	0	0	0	0	0	2.34	149.25	0	151.59	2.34
Customer Before	0	0	0	0	0	0	2.7	13.6	16.3	13.6
TOTAL	2.03	1.8	9.02	54.69	199.06	177.36	436.51	42.97	923.44	544.43

AFTER IMPROVEMENTS

PHASE DETECTED

PHASE INTRODUCED	Planning	Customer	Rqmts. Analysis	Design	Implementation	Test	Formal Test	Customer Before	TOTAL	Leaked
Planning	16.45	2	0	0.51	0	0	0	0	18.96	2.51
Customer	0	3.3	15.4	7	8.12	0	0	0	33.82	30.52
Rqmts. Analysis	0	1	19.32	11.18	7.91	3.2	0	0	42.61	23.29
Design	0	0	0.39	22.49	11.1	1.6	0	0	35.58	13.09
Implementation	0	0	0	0.2	239.4	42	0	2.33	283.93	44.53
Test	0	0	0	0.16	0.06	1.76	0.15	0	2.13	0.37
Formal Test	0	0	0	0	0	0	0.5	0.58	1.08	0.58
Customer Before	0	0	0	0	0	0	0	10.2	10.2	10.2
TOTAL	16.45	6.3	35.11	41.54	266.59	48.56	0.65	13.11	428.31	125.09

Figure 12

Further investigation revealed that the use of criteria was restricting what the reviewers were looking for in the peer reviews. Training was developed to educate the reviewers on how to use the criteria. The criteria became a living document and as defects were found the checklist were updated.

The results showed remarkable improvements. The number of defects introduced before and after improvements were in the same order of magnitude (1947 vs. 1166) so that comparisons could be made between the "before" and "after" states. If you only look at the percentage of defects found in phase, as a lot of organizations do, the results can be misleading. It shows that five of eight phases actually found fewer defects in phase. Analyzing the data this way assumes all defects are created equal (it takes the same amount of effort to fix the defect) and does not take into effect the number of phases the defect leaked.

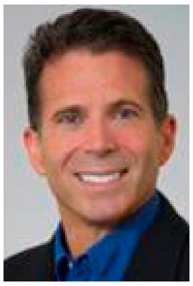
If the defects are transformed into the amount of rework, a completely different profile is observed. In those five phases that found a smaller percent of defects in phase the amount of rework decreased by 75%. Looking at the three phases that accounted for 92% of the rework, the improvements are dramatic. It can be confidently stated that two of the three phases will exceed the goal of finding 80% of defects in phase. The third phase only allowed 1% of the defects to make it to test, whereas before the improvements, 14% made it to test. This reduced the rework from 156 days to a mere 13. Remember, measure what are you trying to improve—is it number of defects or rework?

The bottom line savings exceeded the goal by more than 20%. There was a nominal increase in cost in the early stage but, as can be seen by the graph, the cost of rework leveled off after the implementation phase. This means almost no defects leaked into the testing phase or beyond. Imagine your organization having no defects leak beyond the implementation phase. It can be done!

Disclaimer:

CMMI® and CMM® are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

ABOUT THE AUTHOR



Tom Lienhard is a Sr. Principal Engineer at Raytheon Missile System's Tucson facility and a Six Sigma BlackBelt. Tom has participated in more than 50 CMM® and CMMI® appraisals both in DoD and Commercial environments across North America and Europe and was a member of Raytheon's CMMI Expert Team. He has taught Six Sigma across the globe, and helped various organizations climb the CMM and CMMI maturity levels, including Raytheon Missile System's achievement of CMMI Level 5.

He has received the AlliedSignal Quest for Excellence Award, the Raytheon Technology Award and the Raytheon Excellence in Operations and Quality Award. Tom has a BS in computer science and has worked for Hughes, Raytheon, AlliedSignal, Honeywell and as a consultant for Managed Process Gains.

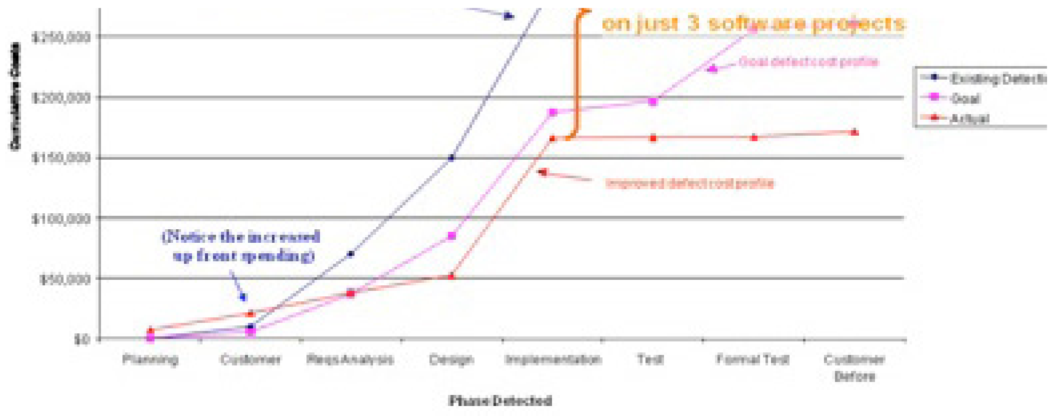


Figure 13

Upcoming Events

Visit <http://www.crosstalkonline.org/events> for an up-to-date list of events.

Annual Computer Security Applications Conference
3-7 December 2012
Orlando, FL
http://www.acsac.org

International Conference on Computing and Information Technology
14-15 Jan 2013
Zurich, Switzerland
http://www.waset.org/conferences/2013/zurich/iccit/

Technology Tools for Today (T3) Conference
11-13 Feb 2013
Miami, FL
http://www.technologytoolsfortoday.com/conference.html

Strata Conference: Making Data Work
26-28 Feb 2013
Santa Clara, CA
http://strataconf.com/strata2013

Software Assurance Forum - March 2013
5-7 March 2013
Gaithersburg, MD
https://buildsecurityin.us-cert.gov/bsi/events/1417-BSI.html

Government Contracting
12-13 March 2013
Washington, DC
http://publiccontractinginstitute.com/events

Conference on Systems Engineering Research
19-22 March 2013
Atlanta, GA
http://cser13.gatech.edu

7th International Symposium on Service Oriented System Engineering
25-28 March 2013
San Francisco, CA
http://sei.pku.edu.cn/conference/sose2013

Symposium of Mobile Cloud, Computing and Service Engineering
25-28 March 2013
Redwood, CA
http://www.engr.sjsu.edu/gaojerry/IEEEMobileCloud2013/index.htm

Software Technology Conference
8-11 April 2013
Salt Lake City, UT
http://www.sstc-online.org

7th Annual IEEE Systems Conference
15-18 April
Orlando, FL
http://ieeesyscon.org

Systems Engineering, Test and Evaluation Conference
29 April – 1 May 2013
Canberra, Australia
http://sapmea.asn.au/conventions/sete2013

IBM Edge 2013
10-14 Jun 2013
Las Vegas, NV
http://www.ibm.com/edge

23rd Annual INCOSE International Symposium
24-27 Jun 2013
Philadelphia, PA
http://www.incose.org/symp2013

Writing Good Software
Lessons Learned The Hard Way

In 1986, I was one of the founders of the Advanced Software Engineering Education and Training (ASEET) team. We were composed of members from all four services, plus civilian contractors. Our mission was to spread the word about software engineering and how it was much more than just programming. ASEET is long gone, but I still teach, present, and consult. After years of working with universities, military organizations and DoD contractors, I started to learn (often the hard way) what the elements of good software engineering really are.

I boil it down to four attributes—software must be reliable, understandable, modifiable, and efficient.

You can easily add other items; affordable, usable, delivered on time, fault tolerant, and more. However, I have found that the above framework covers the attributes I expect in "good" code. When I teach software engineering, I use the above list. I find it so important that I put some form of question about this list on every test I give. This list is not my creation. I got this framework from *Software Engineering with Ada*, by Grady Booch, back in 1983. Mr. Booch was Chief Scientist for Rational Computers, one of the developers of the Unified Modeling Language, and now is with the IBM Thomas J. Watson Research Center, serving as Chief Scientist for Software Engineering. If this list was good enough for him, it is good enough for me. (As a side note, he also taught at the USAF Academy. I brag that I was the last person to hold the "Booch Chair of Software Engineering" when I taught at the Academy. It was not really a formal position. One day, somebody mentioned to me that I was using Grady Booch's old chair and office. The chair was replaced while I was there—so nobody else held the chair after me.)

I want to include 15 lessons I have learned over the years. Some lessons were easy to learn; some are ones I paid dearly for. Some are my own; some are lessons that I was wise enough to learn from others.

1. Always follow the Attributes of Good Software listed above.
2. You need a coding standard and the self-discipline to follow it.
3. Document why, not what.
4. Code as if the next person to maintain your code is a homicidal maniac who knows where you live. (John F. Woods)
5. A software engineering expert is a person who knows enough about what is really going on to be scared. (Adapted from P. J. Plauger)
6. The foolish learn from experience. The wise learn from the experience of others.
7. Want to be a good programmer? Maintain your own code after letting it sit for a few months.
8. Want to be a really good programmer? Maintain somebody else's code (and everybody else's code will always be "bad".)
9. Want to make your code foolproof? Not likely. They are breeding a better quality of foolish users at an amazing rate.
10. Need the code really bad? We can deliver it that way if you do not quit rushing us. (Courtesy of many friends –thanks to Gene, Les, Lindy, and all of the other founding members of the ASEET team. And thanks to all of the members during the 15 years the ASEET existed.)
11. You cannot have too many backups.
12. If you can overload basic operations (such as "+", "-", etc.), then your language values "cool" over maintainability. I do not want to see "x = x+1" and have to debug for hours to find out that it is not adding 1, but instead invoking a user-defined "+" function with bizarre side effects. "DoSomeWeirdOperation(x)" makes debugging and maintenance a lot easier. (Discovered independently by legions of Ada and C++ programmers. In one program I was debugging, I eventually discovered "a < b" was calling a user-defined function, but "a>b" was not. Given 12 programmers for a jury, I thought I could get off with "justifiable homicide.")
13. In many languages "float_num = integer_num1 / integer_num2" does integer division, truncates, and then converts to a float. "y = 3/2;" is always going to be 1, even if "y" is a float. (This is why I like strongly typed languages, which flag it a syntax error. I really learned this the hard way!)
14. Indexing strings and arrays? If you switch from languages that start indexing at 0 to languages that start at 1 (or vice versa) you are going to write loops with an "off by 1" error. (Relearned every time I switch from Ada to C/C++ to Ada…….)
15. Always carry a short extension cord when you travel. When you need to charge your computer, tablet and phone all at the same time, inevitably the single outlet in the hotel room will be behind the TV stand. (Nothing to do with software, but truly a lesson I have learned the hard way.)

E-mail me your own Lessons Learned The Hard Way. I will try and publish them in a later column.

David A. Cook, Ph.D.
Stephen F. Austin State University
cookda@sfasu.edu

Hill Air Force Base is hiring SOFTWARE ENGINEERS AND COMPUTER SCIENTISTS

Exciting and Stable Workloads:

- ★ Joint Mission Planning System
- ★ Battle Control System-Fixed
- ★ Satellite Technology
- ★ Expeditionary Fighting Vehicle
- ★ F-16, F-22, F-35, New Workloads Coming Soon
- ★ Ground Theater Air Control System
- ★ Human Engineering Development

Employee Benefits:

- ★ Health Care Packages
- ★ 10 Paid Holidays
- ★ Paid Sick Leave
- ★ Exercise Time
- ★ Career Coaching
- ★ Tuition Assistance
- ★ Retirement Savings Plans
- ★ Leadership Training

Location, Location, Location:

- ★ 25 minutes from Salt Lake City
- ★ Utah Jazz Basketball
- ★ Three Minor League Baseball Teams
- ★ One Hour from 12 Ski Resorts
- ★ Minutes from Hunting, Fishing, Water Skiing, ATV Trails, Hiking

Like facebook
www.facebook.com/309SoftwareMaintenanceGroup

Contact Us:

Email: *309SMXG.SODO@hill.af.mil*
Phone: (801) 775-5555

U.S. Department of Homeland Security

NAVAIR

309th Software Maintenance Group

CrossTalk thanks the above organizations for providing their support.